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:	Голубев В.С.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	4
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

Main.cpp:

```
#include <iostream>
                        #include <vector>
                        #include "figure.h"
                        #include "pentagon.h"
                        #include "trapeze.h"
                        #include "rhombus.h"
                        int main() {
                            std::vector<figure*> figures;
                            for(;;) {
                                int command = 0;
                                std::cin >> command;
                                if (command == 0) {
                                    break;
                                } else if(command == 1) {
                                    int figure_id;
                                    std::cin >> figure_id;
                                    figure* new_figure;
                                    if (figure_id == 0) {
                                        new_figure = new trapeze(std::cin);
                                    } else if (figure_id == 1) {
                                        new_figure = new rhombus(std::cin);
                                    } else if (figure_id == 2) {
                                        new_figure = new pentagon(std::cin);
                                    } else {
                                        std::cout << "no such a figure\n" <<
                        std::endl;
                                        exit(-1);
                                    }
                                    figures.push_back(new_figure);
                                } else if(command == 2) {
                                    int function_id;
                                    std::cin >> function_id;
                                    if (function_id == 0) {
                                        for(figure* cur_figure: figures) {
                                            cur_figure->print(std::cout);
                                        }
                                    } else if (function_id == 1){
                                        for(figure* cur_figure: figures) {
                                            std::cout << cur_figure->area() <<
                        '\n' << '\n';
                                        }
                                    } else if (function_id == 2) {
                                        for(figure* cur_figure: figures) {
                                            std::cout << cur_figure->center() <<
                        '\n';
```

```
                }
            }
        } else if (command == 3) {
            int id;
            std::cin >> id;
            if (id >= figures.size()) {
                std::cout << "no such an element" << '\n';
                exit(-1);
            }
            delete figures[id];
            figures.erase(figures.begin() + id);
        }
    }
    for (size_t i = 0; i < figures.size(); ++i) {
        delete figures[i];
    }
    return 0;
}
```

Figure.h:

```
#include <iostream>
#include "point.h"
#include <cmath>
#ifndef LAB3_FIGURE_H
#define LAB3_FIGURE_H

struct figure {
   virtual point center() const = 0;
   virtual std::ostream& print(std::ostream &os) const = 0;
   virtual double area() const = 0;
   virtual ~figure() {};
   double VectNorm(point l, point r) const;
   double VectPropX(point l, point r) const;
   double VectPropY(point l, point r) const;
   double ScalProd(point l, point r) const;
   double TrAngle(point a, point b, point c) const;
  double VectProd(point a, point b) const;
};
#endif //OOP_EXERCISE_03_FIGURE_H
```

Figure.cpp:

```
#include "figure.h"
```

```
#include <cmath>

double figure::VectNorm(point l, point r) const {
    point vect = operator-(l, r);
    double res = sqrt(vect.x * vect.x + vect.y * vect.y);
    return res;
}
double figure::VectPropX(point l, point r) const {
    double res;
    if (l.x == 0 || r.x == 0) {
        res = 0;
    } else {
        res = l.x / r.x;
    }
    return res;
}
double figure::VectPropY(point l, point r) const{
    double res;
    if (l.y == 0 || r.y == 0) {
        res = 0;
    } else {
        res = l.y / r.y ;
    }
    return res;
}

double figure::ScalProd(point l, point r) const {
    return std::abs(l.x * r.x + l.y * r.y);
}
double::figure::TrAngle(point a, point b, point c) const {
     point  v1, v2;
     v1 = operator-(a, b);
     v2 = operator-(a, c);
     return std::abs(v1.x * v2.y - v2.x * v1.y) / 2;
}
double::figure::VectProd(point a, point b) const {
       return a.x * b.y – b.x * a.y ;
}
```

Point.h:

```
#include <iostream>
#ifndef LAB3_POINT_H
#define LAB3_POINT_H
struct point {
```

```
    double x, y;
};
point operator+(point l, point r);
point operator-(point l, point r);
point operator*(point p, double a);
point operator/(point p, double a);
std::istream &operator>>(std::istream &is, point &p);
std::ostream &operator<<(std::ostream &os, const point &p);

#endif //LAB3_POINT_H
```

Point.cpp:

```
#include "point.h"
point operator+ (point l, point r) {
    return {l.x + r.x, l.y + r.y};
};
point operator- (point l, point r) {
    return { r.x - l.x, r.y - l.y};
};
point operator* (point p, double a) {
    return {p.x * a, p.y * a};
};
point operator/ (point p, double a) {
    return {p.x / a, p.y / a};
};
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;
}
```

Pentagon.h:

```
  include <cmath>
#include "figure.h"
#ifndef LAB3_PENTAGON_H
#define LAB3_PENTAGON_H
class pentagon : public figure {
public:
    pentagon();
```

```
    pentagon(const point& a, const point& b, const point& c, const point& d,
const point& e);
    pentagon(std::istream& is);
    double area() const override;
    point center() const override;
    std::ostream& print(std::ostream& os) const override;
private:
    point a_;
    point b_;
    point c_;
    point d_;
    point e_;
};

#endif LAB3_PENTAGON_H //PENTAGON
```

Pentagon.cpp:

```
#include "pentagon.h"
#define PI 3.14159265
pentagon::pentagon() : a_{0, 0}, b_{0, 0}, c_{0, 0}, d_{0, 0}, e_{0, 0} {}
pentagon::pentagon(const point &a, const point &b, const point &c, const
point &d, const point &e) : a_(a), b_(b), c_(c), d_(d), e_(e) {}
pentagon::pentagon(std::istream &is) {
    is >> a_ >> b_ >> c_ >> d_>> e_;
}
point pentagon::center() const {
    return point {((a_.x + b_.x + c_.x + d_.x + e_.x) / 5), ((a_.y + b_.y + c_.y +
d_.y + e_.y) / 5) } ;
}
double pentagon::area() const {
        return TrAngle(a_,b_,c_) + TrAngle(c_,d_,e_) + TrAngle(a_,c_,e_);
}
std::ostream& pentagon::print(std::ostream &os) const {
    os << a_ << b_ << c_ << d_ << e_;
}
```

Rhombus.h:

```
#include "figure.h"
#ifndef LAB3_RHOMBUS_H
#define LAB3_RHOMBUS_H
class rhombus : public figure {
public:
    rhombus();
    rhombus(const point& a, const point& b, const point& c, const point& d);
    rhombus(std::istream& is);
```

```
    double area() const override;
    point center() const override;
    std::ostream& print(std::ostream& os) const override;
private:
    point a_;
    point b_;
    point c_;
    point d_;
};
#endif //LAB3_RHOMBUS_H
```

Rhombus.cpp:

```
#include "rhombus.h"
#include <cmath>
rhombus::rhombus() : a_{0, 0}, b_{0, 0}, c_{0, 0}, d_{0, 0} {}
rhombus::rhombus(const point &a, const point &b, const point &c, const point
&d) : a_(a), b_(b), c_(c), d_(d) {};
rhombus::rhombus(std::istream &is) {
    is >> a_ >> b_ >> c_ >> d_;
    if ( VectProd(operator-(a_, b_), operator-(d_, c_)) == 0  &&
         VectProd(operator-(a_, d_), operator-(b_, c_)) == 0 &&
         ScalProd(operator-(c_,a_), operator-(d_,b_)) == 0 ) {
        std::cout << "Correct" << std::endl;

    }
    else {
        std::cout << "Wrong" << std::endl;
        throw 1;


    }
}
double rhombus::area() const {
    return VectNorm(c_, a_) * VectNorm(d_, b_) / 2;
}

point rhombus::center() const {
    return point{((a_.x + b_.x + c_.x + d_.x) / 4), ((a_.y + b_.y + c_.y + d_.y) /
4)};
}

std::ostream &rhombus::print(std::ostream &os) const {
    os << a_ << b_ << c_ << d_ << std::endl;
    return os;
```

```
}
```

Trapeze.h:

```
#include "figure.h"
#ifndef LAB3_TRAPEZE_H
#define LAB3_TRAPEZE_H
class trapeze : public figure {
public:
    trapeze();
    trapeze(const point& a, const point& b, const point& c, const point& d);
    trapeze(std::istream& is);
    double area() const override;
    point center() const override;
    std::ostream& print(std::ostream& os) const override;
private:
    point a_;
    point b_;
    point c_;
    point d_;
};
#endif //LAB3_TRAPEZE_H
```

Trapeze.cpp:

```
#include "trapeze.h"
#include <cmath>

trapeze::trapeze(): a_{0, 0}, b_{0, 0}, c_{0, 0}, d_{0, 0} {}
trapeze::trapeze(const point &a, const point &b, const point &c, const point
&d) : a_{a}, b_{b}, c_{c}, d_{d} {}
trapeze::trapeze(std::istream &is) {
    is >> a_ >> b_ >> c_ >> d_;
    if ( ( VectProd(operator-(a_,b_), operator-(d_,c_)) == 0) ||
         (VectProd(operator-(b_,c_), operator-(a_,d_)) == 0) ) {
        std::cout << "Correct" << std::endl;
    } else {
        std::cout << "Wrong" << std::endl;
        throw 1;
    }
}
double trapeze::area() const {
```

```
   if ( (VectProd(operator-(a_,b_), operator-(c_,d_)) == 0) &&
(VectProd(operator-(b_,c_), operator-(a_,d_)) == 0) ) {
       return fabs((VectProd(operator-(a_,b_), operator-(a_,d_)))) ;
   } else if (VectProd(operator-(a_,b_), operator-(d_,c_)) == 0) {
       return ((VectNorm(a_, b_) + VectNorm(d_, c_)) / 2) * sqrt(
             VectNorm(d_, a_) * VectNorm(d_, a_) - (
                   pow((
                         (pow((VectNorm(d_, c_) - VectNorm(a_, b_)), 2) +
                          VectNorm(d_, a_) * VectNorm(d_, a_) - VectNorm(b_,
c_) * VectNorm(b_, c_)) /
                         (2 * (VectNorm(d_, c_) - VectNorm(a_, b_)))
                      ), 2)
             )
       );

   } else if (VectProd(operator-(b_,c_), operator-(a_,d_)) == 0) {
       return ((VectNorm(b_, c_) + VectNorm(a_, d_)) / 2) * sqrt(
             VectNorm(a_, b_) * VectNorm(a_, b_) - (
                   pow((
                         (pow((VectNorm(a_, d_) - VectNorm(b_, c_)), 2) +
                          VectNorm(a_, b_) * VectNorm(a_, b_) - VectNorm(c_,
d_) * VectNorm(c_, d_)) /
                         (2 * (VectNorm(a_, d_) - VectNorm(b_, c_)))
                      ), 2)
             )
       );
   }
}

point trapeze::center() const {
   return point{((a_.x + b_.x + c_.x + d_.x) / 4), ((a_.y + b_.y + c_.y + d_.y) /
4)};
}

std::ostream &trapeze::print(std::ostream &os) const {
   os << a_ << b_ << c_ << d_ << std::endl;
   return os;
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.14)
project(lab3)

set(CMAKE_CXX_STANDARD 14)
```

add_executable(lab3 point.h point.cpp trapeze.h trapeze.cpp pentagon.h pentagon.cpp rhombus.h rhombus.cpp figure.h figure.cpp main.c

2. Ссылка на репозиторий на GitHub

https://github.com/VSGolubev-bald/oop_exercise_03

2. Набор testcases.

```
0
1 4
6 4
6 0
0 0
1
1
0 2
4 0
2 4
-2 6
1
2
-2 1
-1 1
0 0
-1 -1
-3 0

2
1
2
0
3
1
2
0
3
0
2
0
```

4. Результат выполнения теста.

```
22
12
3.5
1 4
6 4
6 0
0 0
0 2
4 0
2 4
-2 6
-2 1
-1 1
 0 0
-1 -1
-3 0
1 4
6 4
6 0
0 0
-2 1
-1 1
 0 0
-1 -1
-3 0
-2 1
-1 1
 0 0
-1 -1
-3 0
```

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 трапеция , 1 – создается ромб, 2 – создается пятиугольник.

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, написав общие методы для различных фигур: center(), area(), print(), по-разному определенные в самих классах фигур, что позволяет работать с ними в едином интерфейсе, кроме того было изучено такое понятие, как полностью виртуальный метод, который в классе родителе не определен, но определяется в классах наследниках.