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:	Косогоров В.В.
Группа:	М8О-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	10
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

figure.hpp

```
#ifndef FIGURE_H
#define FIGURE_H

class Figure {

protected:
    double x_[4];
    double y_[4];

public:
    Figure() {}

    virtual int IsCorrect() const = 0;
    virtual std::ostream& CalculateCenter(std::ostream& os) const = 0;
    virtual std::ostream& CalculateArea(std::ostream& os) const = 0;
    virtual void Print(std::ostream& os) const = 0;

    virtual ~Figure() {}
};

#endif //FIGURE_H
```

rectangle.hpp

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include <iostream>
#include <cmath>
#include <assert.h>


#include "figure.hpp"

class Rectangle: public Figure {

public:
    Rectangle();
    Rectangle(std::istream& is);

    int IsCorrect() const override;
```

```
    std::ostream& CalculateCenter(std::ostream& os) const override;
    std::ostream& CalculateArea(std::ostream& os) const override;
    void Print(std::ostream& os) const override;

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

    ~Rectangle() {}
};

#endif //RECTANGLE_H
```

rectangle.cpp

```
#include "rectangle.hpp"

Rectangle::Rectangle() {
   for (int i = 0; i < 4; ++i) {
      x_[i] = 0;
      y_[i] = 0;
   }
}

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

int Rectangle::IsCorrect() const {
   double vec1_x = x_[1] - x_[0];
   double vec1_y = y_[1] - y_[0];

   double vec2_x = x_[2] - x_[1];
   double vec2_y = y_[2] - y_[1];

   double vec3_x = x_[3] - x_[0];
   double vec3_y = y_[3] - y_[0];

   double dotProduct1 = vec1_x * vec2_x + vec1_y * vec2_y;
   double dotProduct2 = vec3_x * vec1_x + vec3_y * vec1_y;

   if (dotProduct1 == 0 && dotProduct2 == 0) {
      return 1;
   }
   return 0;
```

```
}

std::ostream& Rectangle::CalculateCenter(std::ostream& os) const {
    double xCenter = (x_[0] + x_[2]) / 2;
    double yCenter = (y_[0] + y_[2]) / 2;
    os << "Center : " << "(" << xCenter << ", " << yCenter << ')' << std::endl;
    return os;
}

std::ostream& Rectangle::CalculateArea(std::ostream& os) const {
    double xHeight = x_[1] - x_[0];
    double yHeight = y_[1] - y_[0];

    double xWidth = x_[2] - x_[1];
    double yWidth = y_[2] - y_[1];

    double area = sqrt(xHeight * xHeight + yHeight * yHeight) * sqrt(xWidth *
xWidth + yWidth * yWidth);

    os << "Area: " << area << std::endl << std::endl;
    return os;
}

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

std::ostream& operator<< (std::ostream &os, const Rectangle& rectangle) {
    os << "Rectangle:" << std::endl;
    os << "Coordinates: " << "A(" << rectangle.x_[0] << ", " << rectangle.y_[0] << "),
";
    os << "B(" << rectangle.x_[1] << ", " << rectangle.y_[1] << "), ";
    os << "C(" << rectangle.x_[2] << ", " << rectangle.y_[2] << "), ";
    os << "D(" << rectangle.x_[3] << ", " << rectangle.y_[3] << ")" << std::endl;

    return os;
}

std::istream& operator>> (std::istream& is, Rectangle& rectangle) {
    is >> rectangle.x_[0] >> rectangle.y_[0];
    is >> rectangle.x_[1] >> rectangle.y_[1];
    is >> rectangle.x_[2] >> rectangle.y_[2];
    is >> rectangle.x_[3] >> rectangle.y_[3];
```

```
    return is;
}
```

square.hpp

```
#ifndef SQUARE_H
#define SQUARE_H

#include <iostream>
#include <cmath>
#include <assert.h>

#include "figure.hpp"

class Square: public Figure {

public:
    Square();
      Square(std::istream& is);

    int IsCorrect() const override;
      std::ostream& CalculateCenter(std::ostream& os) const override;
      std::ostream& CalculateArea(std::ostream& os) const override;
      void Print(std::ostream& os) const override;

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

    ~Square() {}
};

#endif //  SQUARE_H
```

square.cpp

```
#include "square.hpp"

Square::Square() {
    for (int i = 0; i < 4; ++i) {
        x_[i] = 0;
        y_[i] = 0;
    }
}
```

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

int Square::IsCorrect() const {
   double vec1_x = x_[1] - x_[0];
   double vec1_y = y_[1] - y_[0];

   double vec2_x = x_[2] - x_[1];
   double vec2_y = y_[2] - y_[1];

   double vec3_x = x_[3] - x_[0];
   double vec3_y = y_[3] - y_[0];

   double dotProduct1 = vec1_x * vec2_x + vec1_y * vec2_y;
   double dotProduct2 = vec3_x * vec1_x + vec3_y * vec1_y;

   double vec1_length = sqrt(vec1_x * vec1_x + vec1_y * vec1_y);
   double vec2_length = sqrt(vec2_x * vec2_x + vec2_y * vec2_y);

   if (dotProduct1 == 0 && dotProduct2 == 0 && vec1_length == vec2_length) {
      return 1;
   }
   return 0;
}

std::ostream& Square::CalculateCenter(std::ostream& os) const {
   double xCenter = (x_[0] + x_[2]) / 2;
   double yCenter = (y_[0] + y_[2]) / 2;

   os << "Center : " << "(" << xCenter << ", " << yCenter << ')' << std::endl;
   return os;
}

std::ostream& Square::CalculateArea(std::ostream& os) const {
   double vecX = x_[1] - x_[0];
   double vecY = y_[1] - y_[0];

   double area = vecX * vecX + vecY * vecY;

   os << "Area: " << area << std::endl << std::endl;
   return os;
}
```

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

std::ostream& operator<< (std::ostream &os, const Square& square) {
    os << "Square:" << std::endl;
    os << "Coordinates: " << "A(" << square.x_[0] << ", " << square.y_[0] << "), ";
    os << "B(" << square.x_[1] << ", " << square.y_[1] << "), ";
    os << "C(" << square.x_[2] << ", " << square.y_[2] << "), ";
    os << "D(" << square.x_[3] << ", " << square.y_[3] << ")" << std::endl;

    return os;
}

std::istream& operator>> (std::istream& is, Square& square) {
    is >> square.x_[0] >> square.y_[0];
    is >> square.x_[1] >> square.y_[1];
    is >> square.x_[2] >> square.y_[2];
    is >> square.x_[3] >> square.y_[3];

    return is;
}
```

trapezoid.hpp

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

#include <iostream>
#include <cmath>
#include <assert.h>


#include "figure.hpp"

class Trapezoid: public Figure {

public:
    Trapezoid();
      Trapezoid(std::istream& is);

    int IsCorrect() const override;
      std::ostream& CalculateCenter(std::ostream& os) const override;
      std::ostream& CalculateArea(std::ostream& os) const override;
```

```
    void Print(std::ostream& os) const override;

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

  ~Trapezoid() {}
};

#endif // TRAPEZOID_H
```

trapezoid.cpp

```
 #include "trapezoid.hpp"

Trapezoid::Trapezoid() {
  for (int i = 0; i < 4; ++i) {
    x_[i] = 0;
    y_[i] = 0;
  }
}

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

int Trapezoid::IsCorrect() const {
  double vec1_x = x_[3] - x_[0];
  double vec1_y = y_[3] - y_[0];

  double vec2_x = x_[2] - x_[1];
  double vec2_y = y_[2] - y_[1];

  double vec3_x = x_[1] - x_[0];
  double vec3_y = y_[1] - y_[0];

  double vec4_x = x_[2] - x_[3];
  double vec4_y = y_[2] - y_[3];

  if ((vec1_x / vec2_x == vec1_y / vec2_y) || (vec3_x / vec4_x == vec3_y / vec4_y)
|| //отношение соответствующих координат
       (vec1_x == 0 && vec2_x == 0) || (vec1_y == 0 && vec2_y == 0) || (vec3_x
== 0 && vec4_x == 0) || (vec3_y == 0 && vec4_y == 0)) {
    return 1;
  }
```

```
    return 0;
}

std::ostream& Trapezoid::CalculateCenter(std::ostream& os) const {
    double xCenter = 0;
    double yCenter = 0;

    for (int i = 0; i < 4; ++i) {
        xCenter += x_[i];
        yCenter += y_[i];
    }

    xCenter /= 4;
    yCenter /= 4;

    os << "Center : " << "(" << xCenter << ", " << yCenter << ')' << std::endl;

    return os;
}

std::ostream& Trapezoid::CalculateArea(std::ostream& os) const {
    double area1 = 0.5 * abs((x_[2] - x_[1]) * (y_[3] - y_[1]) - (x_[3] - x_[1]) * (y_[2]
- y_[1]));
    double area2 = 0.5 * abs((x_[0] - x_[1]) * (y_[3] - y_[1]) - (x_[3] - x_[1]) * (y_[0]
- y_[1]));

    double area = area1 + area2;

    os << "Area: " << area << std::endl << std::endl;

    return os;
}

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

std::ostream& operator<< (std::ostream &os, const Trapezoid& trapezoid) {
    os << "Trapezoid:" << std::endl;
    os << "Coordinates: " << "A(" << trapezoid.x_[0] << ", " << trapezoid.y_[0] << "),
";
    os << "B(" << trapezoid.x_[1] << ", " << trapezoid.y_[1] << "), ";
    os << "C(" << trapezoid.x_[2] << ", " << trapezoid.y_[2] << "), ";
    os << "D(" << trapezoid.x_[3] << ", " << trapezoid.y_[3] << ")" << std::endl;
```

```
    return os;
}

std::istream& operator>> (std::istream& is, Trapezoid& trapezoid) {
    is >> trapezoid.x_[0] >> trapezoid.y_[0];
    is >> trapezoid.x_[1] >> trapezoid.y_[1];
    is >> trapezoid.x_[2] >> trapezoid.y_[2];
    is >> trapezoid.x_[3] >> trapezoid.y_[3];

    return is;
}
```

main.cpp:

```
#include <iostream>
#include <vector>
#include <cmath>

#include "figure.hpp"
#include "rectangle.hpp"
#include "square.hpp"
#include "trapezoid.hpp"

int main(void) {
    std::vector<Figure*> figures;
    int input;

    while (true) {
        std::cout << "==================================" << std::endl;
        std::cout << "Available commands:" << std::endl;
        std::cout << "0. Exit" << std::endl;
        std::cout << "1. Add a figure via id" << std::endl;
        std::cout << "2. Print info for every figure" << std::endl;
        std::cout << "3. Remove a figure via id" << std::endl << std::endl;

        std::cin >> input;

        if (input == 0) {
            break;
        }

        if (input > 3) {
            std::cout << "ERROR: invalid command" << std::endl << std::endl;
            continue;
```

```
}

switch(input) {
   case 1:
      int figureID;
      std::cout << "Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): ";
      std::cin >> figureID;

      if (figureID < 1 || figureID > 3) {
         std::cout << "ERROR: invalid id" << std::endl;
         continue;
      }

      std::cout << "Enter 4 (x, y) points in a sequence" << std::endl;
      Figure* newFigure;

      switch (figureID) {
         case 1:
            newFigure = new Square(std::cin);
            figures.push_back(newFigure);
         break;

         case 2:
            newFigure = new Rectangle(std::cin);
            figures.push_back(newFigure);

         break;

         case 3:
            newFigure = new Trapezoid(std::cin);
            figures.push_back(newFigure);
         break;
      }
   break;

   case 2:
      if (figures.size() == 0) {
         std::cout << "No figures to display" << std::endl << std::endl;
      } else {
         int id = 0;
         for (Figure* currentFigure : figures) {
            std::cout << "ID: " << id << std::endl;
            currentFigure->Print(std::cout);
            currentFigure->CalculateCenter(std::cout);
            currentFigure->CalculateArea(std::cout);
```

```
                ++id;
            }
        }
        break;

        case 3:
            size_t id;
            std::cout << "Enter a figure id: ";
            std::cin >> id;

            if (id > figures.size() - 1) {
                std::cout << "ERROR: invalid id" << std::endl << std::endl;
            } else {
                delete figures[id];
                figures.erase(figures.begin() + id);
            }
        break;
      }
   }

   for (size_t i = 0; i < figures.size(); ++i) {
        delete figures[i];
    }
}
```

CmakeLists.txt:

```
cmake_minimum_required(VERSION 3.5)

project(lab3)

add_executable(lab3
  main.cpp
  rectangle.cpp
  square.cpp
  trapezoid.cpp
)

set_property(TARGET lab3 PROPERTY CXX_STANDARD 11)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -g")
```

2. Ссылка на репозиторий на GitHub.

https://github.com/vladiq/oop_exercise_03

3. Набор testcases.

test_01.test:

```
1
1
0 0
1 0
1 1
0 1
2
1
2
1.5 1.5
0.5 2.5
2.5 4.5
3.5 3.5
2
1
3
-1 0
0 2
2 2
3 0
2
3
0
```

test_02.test:

```
1
1
0 0
0 0
1 1
1 1
2
1
2
1000 10
1000 1000
1000 100000
1000 1000000
```

2
1
3
-23121 2423
34312 2323
131232 12312
3231 312312
2
3
0

test_03.test:

1
1
0 0
0 0
0 0
0 0
2
3
4
0
0

test_04.test:

1
1
1 -1
3 -3
5 -1
3 1
1
2
1 0
0 1
999 1000
1000 999
2
3
0

4. Результаты выполнения тестов.

…/prog_3_sem/oop_labs/lab_03 cat testcases/test_01.test

```
1
1
0 0
1 0
1 1
0 1
2
1
2
1.5 1.5
0.5 2.5
2.5 4.5
3.5 3.5
2
1
3
-1 0
0 2
2 2
3 0
2
3
0
```

…/prog_3_sem/oop_labs/lab_03 ./lab3 < testcases/test_01.test

```
==================================
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Square created

==================================
Available commands:
0. Exit
```

1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

ID: 0
Square:
Coordinates: A(0, 0), B(1, 0), C(1, 1), D(0, 1)
Center : (0.5, 0.5)
Area: 1

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Rectangle created

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

ID: 0
Square:
Coordinates: A(0, 0), B(1, 0), C(1, 1), D(0, 1)
Center : (0.5, 0.5)
Area: 1

ID: 1
Rectangle:
Coordinates: A(1.5, 1.5), B(0.5, 2.5), C(2.5, 4.5), D(3.5, 3.5)
Center : (2, 3)
Area: 4

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area

4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Trapezoid created

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

ID: 0
Square:
Coordinates: A(0, 0), B(1, 0), C(1, 1), D(0, 1)
Center : (0.5, 0.5)
Area: 1

ID: 1
Rectangle:
Coordinates: A(1.5, 1.5), B(0.5, 2.5), C(2.5, 4.5), D(3.5, 3.5)
Center : (2, 3)
Area: 4

ID: 2
Trapezoid:
Coordinates: A(-1, 0), B(0, 2), C(2, 2), D(3, 0)
Center : (1, 1)
Area: 6

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Overall area: 11

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

…/prog_3_sem/oop_labs/lab_03 cat testcases/test_02.test

```
1
1
0 0
0 0
1 1
1 1
2
1
2
1000 10
1000 1000
1000 100000
1000 1000000
2
1
3
-23121 2423
34312 2323
131232 12312
3231 312312
2
3
0
```

…/prog_3_sem/oop_labs/lab_03 ./lab3 < testcases/test_02.test

```
==================================
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Wrong sequence or the sides are not equal and/or parallel

==================================
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id
```

No figures to display

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Wrong sequence or the sides are not parallel

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

No figures to display

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Wrong sequence or two opposite sides are not parallel

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

No figures to display

==========
Available commands:

0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

No figures to calculate the area for

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

…/prog_3_sem/oop_labs/lab_03 cat testcases/test_03.test

1
1
0 0
0 0
0 0
0 0
2
3
4
0
0

…/prog_3_sem/oop_labs/lab_03 ./lab3 < testcases/test_03.test
==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Square created

==========
Available commands:

0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

ID: 0
Square:
Coordinates: A(0, 0), B(0, 0), C(0, 0), D(0, 0)
Center : (0, 0)
Area: 0

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Overall area: 0

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id: ==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

…/prog_3_sem/oop_labs/lab_03 cat testcases/test_04.test

1
1
1 -1
3 -3
5 -1
3 1
1
2

1 0
0 1
999 1000
1000 999
2
3
0

…/prog_3_sem/oop_labs/lab_03 ./lab3 < testcases/test_04.test
==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Square created

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Enter a figure id (1 - square, 2 - rectangle, 3 - trapezoid): Enter 4 (
x, y) points in a sequence
Rectangle created

==========
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

ID: 0
Square:
Coordinates: A(1, -1), B(3, -3), C(5, -1), D(3, 1)
Center : (3, -1)
Area: 8

ID: 1

```
Rectangle:
Coordinates: A(1, 0), B(0, 1), C(999, 1000), D(1000, 999)
Center : (500, 500)
Area: 1998

==================================
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id

Overall area: 2006

==================================
Available commands:
0. Exit
1. Add a figure via id
2. Print info for every figure
3. Print the overall area
4. Remove a figure via id
```

5. Объяснение результатов работы программы.

1) Пользователю предоставляется 4 опции: задать фигуру (квадрат, прямоугольник или трапецию), вывести информацию для каждой фигуры (координаты точек, площадь и геометрический центр), вывести общую площадь всех фигур и удалить фигуру по индексу.
2) Перед занесением фигур в вектор каждая фигура проверяется. У квадрата проверяется перпендикулярность и равенство сторон, у прямоугольника — перпендикулярность сторон, у трапеции — параллельность двух противоположных сторон. После чего указатель на созданную фигуру заносится в вектор figures.
3) Вывод информации о всех фигурах производится с помощью цикла. Поочередно перебираются все элементы вектора figures, и с помощью метода Print() выводятся координаты, площадь и геометрический центр каждой из фигур.
4) Общая площадь фигур находится посредством суммирования результата работы метода CalculateArea() для всех фигур вектора.
5) Если пользователь вводит «0», то считывание завершается, а все фигуры удаляются из памяти с помощью delete.

6. Вывод.

Выполняя данную лабораторную, я получил навыки работы с производными классами и виртуальными функциями, познакомился с range-based for циклом и функцией assert(). В ходе работы я создал базовый класс и 3 производных от него класса, которые посредством override методов переопределяли виртуальные методы базового класса.