Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчёт по Лабораторной работе №3
“Наследование, Полиморфизм“
по курсу “Объектно-Объективное Программирование“
III Семестр

Студент:	Катермин В.С.
Группа:	М8О-208Б-18
Преподава-тель:	Журавлёв А.А.
Оценка:	
Дата:	**14.10.19**

1. **Тема**: Наследование, Полиморфизм в C++.
2. **Код программы:**

point.h

```
#ifndef D_POINT_H
#define D_POINT_H

#include <iostream>

struct point{
   double x,y;
};

std::ostream& operator << (std::ostream& os, const point& p);

#endif //D_POINT_H
```

point.cpp

```
#include "point.h"

std::ostream& operator << (std::ostream& os, const point& p){
   return os << p.x << " " << p.y;
}
```

figure.h

```
#ifndef D_FIGURE_H
#define D_FIGURE_H

#include <iostream>

#include "point.h"

struct figure{
   virtual point center() = 0;
   virtual double area() = 0;
   virtual void print(std::ostream& os) = 0;

   virtual ~figure() {};

//std::ostream& operator << (std::ostream& os, const figure& f);
};

#endif //D_FIGURE_H
```

square.h

```
#ifndef D_SQUARE_H
#define D_SQUARE_H

#include "figure.h"

struct square : figure{
  square(const point& p1, const point& p2, const point& p3, const point& p4);
   square(std::istream& is);

   point center() override;
   double area() override;
   void print(std::ostream& os) override;

private:
   point p1_, p2_, p3_, p4_;
};
```

```
#endif //D_SQUARE_H
```

square.cpp

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

square::square(const point& p1, const point& p2, const point& p3, const point& p4):
  p1_(p1), p2_(p2), p3_(p3), p4_(p4) {}

square::square(std::istream& is) {
  is >> p1_.x >> p1_.y >> p2_.x >> p2_.y >> p3_.x >> p3_.y >> p4_.x >> p4_.y;
    assert(((p2_.x - p1_.x)*(p4_.x - p1_.x))+((p2_.y - p1_.y)*(p4_.y - p1_.y)) == 0);
    assert(((p3_.x - p2_.x)*(p1_.x - p2_.x))+((p3_.y - p2_.y)*(p1_.y - p2_.y)) == 0);
    assert(((p4_.x - p3_.x)*(p2_.x - p3_.x))+((p4_.y - p3_.y)*(p2_.y - p3_.y)) == 0);
    assert((p2_.x - p1_.x) == (p1_.y - p4_.y));
    assert((p3_.x - p2_.x) == (p2_.y - p1_.y));
    assert((p4_.x - p3_.x) == (p3_.y - p2_.y));
}

point square::center() {
  return {(p1_.x + p2_.x + p3_.x + p4_.x) * 0.25, (p1_.y + p2_.y + p3_.y + p4_.y) * 0.25};
}

double square::area() {
  const double ds = p1_.x - p2_.x;
  return std::abs(ds * ds);
}

void square::print(std::ostream& os){
  os << "Square [" << p1_ << "] [" << p2_ << "] [" << p3_ << "] [" << p4_ << "]";
}
```

rectangle.h

```
#ifndef D_RECTANGLE_H
#define D_RECTANGLE_H

#include "figure.h"

struct rectangle : figure{
  rectangle(const point& p1, const point& p2, const point& p3, const point& p4);
    rectangle(std::istream& is);

    point center() override;
    double area() override;
    void print(std::ostream& os) override;

private:
    point p1_, p2_, p3_, p4_;
};

#endif //D_RECTANGLE_H
```

rectangle.cpp

```
#include "rectangle.h"
#include <cmath>
#include <cassert>
//#include <algorythm>

rectangle::rectangle(const point& p1, const point& p2, const point& p3, const point& p4):
```

```
  p1_(p1), p2_(p2), p3_(p3), p4_(p4) {}

rectangle::rectangle(std::istream& is){
    is >> p1_.x >> p1_.y >> p2_.x >> p2_.y >> p3_.x >> p3_.y >> p4_.x >> p4_.y;
    assert(((p2_.x - p1_.x)*(p4_.x - p1_.x))+((p2_.y - p1_.y)*(p4_.y - p1_.y)) == 0);
    assert(((p3_.x - p2_.x)*(p1_.x - p2_.x))+((p3_.y - p2_.y)*(p1_.y - p2_.y)) == 0);
    assert(((p4_.x - p3_.x)*(p2_.x - p3_.x))+((p4_.y - p3_.y)*(p2_.y - p3_.y)) == 0);
}

point rectangle::center() {
  return {(p1_.x + p2_.x + p3_.x + p4_.x) * 0.25, (p1_.y + p2_.y + p3_.y + p4_.y) * 0.25};
}

double rectangle::area() {
    const double dx = p1_.x - p2_.x;
    const double dy = p1_.y - p4_.y;
    return std::abs(dx * dy);
}

void rectangle::print(std::ostream& os) {
  os << "Rectangle [" << p1_ << "] [" << p2_ << "] [" << p3_ << "] [" << p4_ << "]";
}
```

trapeze.h

```
#ifndef D_TRAPEZE_H
#define D_TRAPEZE_H

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

struct trapeze : figure{
  trapeze(const point& p1, const point& p2, const point& p3, const point& p4);
    trapeze(std::istream& is);

    point center() override;
    double area() override;
    void print(std::ostream& os) override;

private:
    point p1_, p2_, p3_, p4_;
};

#endif //D_TRAPEZE_H
```

trapeze.cpp

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

trapeze::trapeze(const point& p1, const point& p2, const point& p3, const point& p4):
  p1_(p1), p2_(p2), p3_(p3), p4_(p4) {}

trapeze::trapeze(std::istream& is){
    is >> p1_.x >> p1_.y >> p2_.x >> p2_.y >> p3_.x >> p3_.y >> p4_.x >> p4_.y;
    assert(((p2_.x - p1_.x)*(p4_.y - p3_.y)) == ((p4_.x - p3_.x)*(p2_.y - p1_.y)));
}

point trapeze::center() {
    return {(p1_.x + p2_.x + p3_.x + p4_.x) * 0.25, (p1_.y + p2_.y + p3_.y + p4_.y) * 0.25};
    //return {((p1_.x + p4_.x)* 0.5), ((p1_.y + p2_.y) * 0.5)};
}
```

```
double trapeze::area() {
  const double l1 = p2_.x - p1_.x;
  const double l2 = p4_.x - p3_.x;
  const double lh = ((p4_.x - p1_.x)*(p4_.x - p3_.x)+(p4_.y - p1_.y)*(p4_.y - p3_.y))/sqrt((p4_.x - p3_.x)*(p4_.x - p3_.x)+(p4_.y - p3_.y)*(p4_.y - p3_.y));
  const double h = sqrt((p4_.x - p1_.x)*(p4_.x - p1_.x)+(p4_.y - p1_.y)*(p4_.y - p1_.y)-lh*lh);
  //std::cout << (p4_.x - p1_.x)*(p4_.x - p3_.x) << " "  << (p4_.y - p1_.y)*(p4_.y - p3_.y) << " " << lh << " " << h << std::endl;
  return std::abs(l1 * l2 * h * 0.5);
}

void trapeze::print(std::ostream& os) {
  os << "Trapeze [" << p1_ << "] [" << p2_ << "] [" << p3_ << "] [" << p4_ << "]";
}
```

lab3.cpp

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

#include "square.h"
#include "rectangle.h"
#include "trapeze.h"

int main(){
    std::vector<figure*> figures;
    for (;;){
        int command;
        std::cin >> command;
        if (command == 0){
            break;
        }else if (command == 1){
            int figure_type;
            std::cin >> figure_type;
            figure* ptr;
            if (figure_type == 0){
                ptr = new square(std::cin);
            }else if (figure_type == 1){
                ptr = new rectangle(std::cin);
            } else {
             ptr = new trapeze(std::cin);
            }
            figures.push_back(ptr);
        }else if (command == 2){
            int id;
            std::cin >> id;
            delete figures[id];
            figures.erase(figures.begin() + id);
        }else if (command == 3){
            std::cout << "Centers:\n";
            for (figure* ptr: figures){
                std::cout << ptr->center() << std::endl;
            }
        }else if (command == 4){
            std::cout << "Areas:\n";
            for (figure* ptr: figures){
                std::cout << ptr->area() << std::endl;
            }
        }else if (command == 5){
            std::cout << "Figures:\n";
            for (figure* ptr: figures){
                ptr->print(std::cout);
                std::cout << std::endl;
            }
```

```
        }
    }
    for (figure* ptr: figures){
        delete ptr;
    }
}
```

Makefile

```
project(lab3)

add_executable(lab3
  ./lab3.cpp
  ./point.cpp
  ./square.cpp
  ./rectangle.cpp
  ./trapeze.cpp)

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

3. **Ссылка на репозиторий:**
   https://github.com/GitGood2000/oop_exercise_03

4. **Набор testcases:**

test_00.test
```
1
0
0 2 2 2 2 0 0 0
1
1
1 7 4 7 4 2 1 2
1
2
2 4 6 4 7 1 1 1
3
4
5
0
```

test_00.result
```
Centers:
1 1
2.5 4.5
4 2.5
Areas:
4
15
36
Figures:
Square [0 2] [2 2] [2 0] [0 0]
Rectangle [1 7] [4 7] [4 2] [1 2]
Trapeze [2 4] [6 4] [7 1] [1 1]
```

test_01.test
```
1
0
0 2 2 3 3 1 1 0
1
0
1 3 4 6 7 3 4 0
```

1
1
0 2 4 4 5 2 1 0
1
1
1 3 7 5 8 2 2 0
1
2
0 3 3 5 6 5 0 1
1
2
1 5 7 3 5 1 2 2
3
4
5
2
1
5
2
2
5
2
3
5
0

test_01.result

Centers:
1.5 1.5
4 3
2.5 2
4.5 2.5
2.25 3.5
3.75 2.75
Areas:
4
9
8
18
14.9769
22.7684
Figures:
Square [0 2] [2 3] [3 1] [1 0]
Square [1 3] [4 6] [7 3] [4 0]
Rectangle [0 2] [4 4] [5 2] [1 0]
Rectangle [1 3] [7 5] [8 2] [2 0]
Trapeze [0 3] [3 5] [6 5] [0 1]
Trapeze [1 5] [7 3] [5 1] [2 2]
Figures:
Square [0 2] [2 3] [3 1] [1 0]
Rectangle [0 2] [4 4] [5 2] [1 0]
Rectangle [1 3] [7 5] [8 2] [2 0]
Trapeze [0 3] [3 5] [6 5] [0 1]
Trapeze [1 5] [7 3] [5 1] [2 2]
Figures:
Square [0 2] [2 3] [3 1] [1 0]
Rectangle [0 2] [4 4] [5 2] [1 0]
Trapeze [0 3] [3 5] [6 5] [0 1]
Trapeze [1 5] [7 3] [5 1] [2 2]
Figures:
Square [0 2] [2 3] [3 1] [1 0]
Rectangle [0 2] [4 4] [5 2] [1 0]
Trapeze [0 3] [3 5] [6 5] [0 1]

5. **Результаты выполнения тестов:**

```
user@PSB133S01ZFH:~/3sem_projects/oop_exercise_03/tests$ bash test.sh ../build/lab3
Test test_00.test: SUCCESS
Test test_01.test: SUCCESS
```

6. **Объяснение результатов работы программы:**
   1) Программа создаёт вектор — там мы будем хранить данные о фигурах;
   2) Каждая структура геометрических фигур (square, rectangle, trapeze) является потомком структуры figure (центр тяжести, площадь, вывод данных о фигуре), которая является потомком структуры point (координаты x и y);
   3) Программа выполняет команды в виде чисел:
      A) 0 – выход;
      B) 1 — создание элемента и добавление в него данных об определённой фигуре (4 вершины):
         I. 0 — квадрат;
         II. 1 — прямоугольник;
         III. 2 – трапеция.
      C) 2 — удаление элемента по его индексу;
      D) 3 — вывести координаты центров тяжести всех элементов
      E) 4 — вывести площади всех элементов
      F) 5 — вывести введённые данные всех элементов
   4) При выходе программа удаляет вектор со всеми элементами.

7. **Вывод:** 1) Ознакомились с наследованием и полиморфизмом в C++ и усвоили навык работы с ними; 2) Написана программа, производящая операции с помощью наследования и полиморфизма.