Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа**

**по курсу «Объектно-ориентированное программирование»**

**III Семестр**

**Задание 3  
Вариант 8**

**Наследование, полиморфизм**

|  |  |
| --- | --- |
| Студент: | Жерлыгин М.А |
| Группа: | М8О-208Б-18 |
| Преподаватель: | Журалвев А.А. |
|  |  |
| Оценка: |  |
| Дата: |  |

# 1. Код программы на языке С++

**point.h:**

#ifndef D\_POINT\_H\_

#define D\_POINT\_H\_

#include <istream>

#include <ostream>

**class** **Point** {

**public:**

**double** x, y;

Point();

Point(**double** a, **double** b);

//Point& operator=(const Point& other);

Point **operator**+(**const** Point& other);

Point **operator**-(**const** Point& other);

Point **operator**/(**const** **double** num);

~Point() = **default**;

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

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

};

#endif //D\_POINT\_H\_

**point.cpp:**

#include "point.h"

#include <cmath>

Point::Point(): x(**0**), y(**0**) {

}

Point::Point(**double** a, **double** b): x(a), y(b) {

}

/\*Point& Point::operator=(const Point& other) {

this->x = other.x;

this->y = other.y;

return \*this;

}\*/

Point Point::**operator**+(**const** Point& other) {

Point result;

result.x = **this**->x + other.x;

result.y = **this**->y + other.y;

**return** result;

}

Point Point::**operator**-(**const** Point& other) {

Point result;

result.x = **this**->x - other.x;

result.y = **this**->y - other.y;

**return** result;

}

Point Point::**operator**/(**const** **double** num) {

Point result;

result.x = **this**->x / num;

result.y = **this**->y / num;

**return** result;

}

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

**return** is >> p.x >> p.y;

}

std::ostream& **operator**<< (std::ostream& os, **const** Point& p) {

**return** os << "(" << p.x << ", " << p.y << ")" << std::endl;

}

**figure.h:**

#ifndef D\_FIGURE\_H\_

#define D\_FIGURE\_H\_

#include <iostream>

#include "point.h"

**class** **Figure** {

**public:**

**virtual** Point center() **const** = **0**;

**virtual** **double** area() **const** = **0**;

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

**virtual** ~Figure() = **default**;

};

std::ostream& **operator**<< (std::ostream& os, **const** Figure& f);

#endif //D\_FIGURE\_H\_

**figure.cpp:**

#include "figure.h"

std::ostream& **operator**<< (std::ostream& os, **const** Figure& f) {

f.print(os);

**return** os;

}

**octagon.h:**

#ifndef D\_OCTAGON\_H\_

#define D\_OCTAGON\_H\_

#include "figure.h"

#include <cmath>

**class** **Octagon** : **public** Figure {

**private:**

Point coordinate[**8**];

**public:**

Octagon();

Octagon(std::istream& is);

Point center() **const** override;

**double** area() **const** override;

**void** print(std::ostream& os) **const** override;

};

#endif //D\_OCTAGON\_H\_

**octagon.cpp:**

#include <iostream>

#include <cmath>

#include "octagon.h"

Octagon::Octagon() {

**for**(**int** i = **0**; i < **8**; i++) {

coordinate[i].x = **0.0**;

coordinate[i].y = **0.0**;

}

}

Octagon::Octagon(std::istream& is) {

**for**(**int** i = **0**; i < **8**; i++) {

is >> coordinate[i];

}

}

**double** Octagon::area() **const** {

**double** result = **0**;

**for**(**int** i = **0**; i < **7**; i++) {

result += (coordinate[i].x \* coordinate[i+**1**].y) - (coordinate[i+**1**].x \* coordinate[i].y);

}

result = std::abs(result + (coordinate[**7**].x \* coordinate[**0**].y) - (coordinate[**0**].x \* coordinate[**7**].y));

**return** result / **2.0**;

}

Point Octagon::center() **const** {

Point result;

**for**(**int** i = **0**; i < **8**; i++) {

result = result + coordinate[i];

}

**return** result / **8.0**;

}

**void** Octagon::print(std::ostream& os) **const** {

std::cout << "Octagon coordinates:" << std::endl;

os << **this**->coordinate[**0**];

os << **this**->coordinate[**1**];

os << **this**->coordinate[**2**];

os << **this**->coordinate[**3**];

os << **this**->coordinate[**4**];

os << **this**->coordinate[**5**];

os << **this**->coordinate[**6**];

os << **this**->coordinate[**7**];

}

**triangle.h:**

#ifndef D\_TRIANGLE\_H\_

#define D\_TRIANGLE\_H\_

#include "figure.h"

**class** **Triangle** : **public** Figure {

**public:**

Point coordinate[**3**];

Triangle();

Triangle(std::istream& is);

Point center() **const** override;

**double** area() **const** override;

**void** print(std::ostream& os) **const** override;

};

#endif //D\_TRIANGLE\_H\_

**triangle.cpp:**

#include <iostream>

#include <cmath>

#include "triangle.h"

Triangle::Triangle() {

//coordinate = new Point[3];

**for**(**int** i = **0**; i < **3**; i++) {

coordinate[i].x = **0.0**;

coordinate[i].y = **0.0**;

}

}

Triangle::Triangle(std::istream& is) {

//coordinate = new Point[3];

**for**(**int** i = **0**; i < **3**; i++) {

is >> coordinate[i];

}

**double** AB, BC, AC;

AB = sqrt(pow(coordinate[**1**].x - coordinate[**0**].x, **2**) + pow(coordinate[**1**].y - coordinate[**0**].y, **2**));

BC = sqrt(pow(coordinate[**2**].x - coordinate[**1**].x, **2**) + pow(coordinate[**2**].y - coordinate[**1**].y, **2**));

AC = sqrt(pow(coordinate[**2**].x - coordinate[**0**].x, **2**) + pow(coordinate[**2**].y - coordinate[**0**].y, **2**));

**if**(AB + BC <= AC || AB + AC <= BC || BC + AC <= AB) **throw** std::logic\_error("This is not Triange");

}

Point Triangle::center() **const** {

Point result;

**for**(**int** i = **0**; i < **3**; i++) {

result = result + coordinate[i];

}

**return** result / **3.0**;

}

**double** Triangle::area() **const** {

**return** fabs(((coordinate[**0**].x - coordinate[**2**].x) \* (coordinate[**1**].y - coordinate[**2**].y) - (coordinate[**1**].x - coordinate[**2**].x) \* (coordinate[**0**].y - coordinate[**2**].y)) / **2**);

}

**void** Triangle::print(std::ostream& os) **const** {

std::cout << "Triangle coordinates" << std::endl;

os << Point(coordinate[**0**].x, coordinate[**0**].y) << "**\n**" << Point(coordinate[**1**].x, coordinate[**1**].y) << "**\n**" << Point(coordinate[**2**].x, coordinate[**2**].y) << std::endl;

}

**square.h:**

#ifndef D\_Square\_H\_

#define D\_Square\_H\_

#include "figure.h"

**struct** Square : **public** Figure {

**private:**

Point coordinate[**4**];

**public:**

Square();

Square(std::istream& is);

Point center() **const** override;

**double** area() **const** override;

**void** print(std::ostream& os) **const** override;

};

#endif // D\_Square\_H\_

**square.cpp:**

#include <iostream>

#include "square.h"

#include <cmath>

#include <algorithm>

Square::Square() {

**for**(**int** i = **0**; i < **4**; i++) {

coordinate[i].x = **0.0**;

coordinate[i].y = **0.0**;

}

}

Square::Square(std::istream& is) {

**double** a, b, c, d;

is >> coordinate[**0**];

is >> coordinate[**1**];

is >> coordinate[**2**];

is >> coordinate[**3**];

a = sqrt((coordinate[**1**].x - coordinate[**0**].x)\*(coordinate[**1**].x - coordinate[**0**].x) + (coordinate[**1**].y - coordinate[**0**].y)\*(coordinate[**1**].y - coordinate[**0**].y));

b = sqrt((coordinate[**2**].x - coordinate[**1**].x)\*(coordinate[**2**].x - coordinate[**1**].x) + (coordinate[**2**].y - coordinate[**1**].y)\*(coordinate[**2**].y - coordinate[**1**].y));

c = sqrt((coordinate[**3**].x - coordinate[**2**].x)\*(coordinate[**3**].x - coordinate[**2**].x) + (coordinate[**3**].y - coordinate[**2**].y)\*(coordinate[**3**].y - coordinate[**2**].y));

d = sqrt((coordinate[**0**].x - coordinate[**3**].x)\*(coordinate[**0**].x - coordinate[**3**].x) + (coordinate[**0**].y - coordinate[**3**].y)\*(coordinate[**0**].y - coordinate[**3**].y));

**double** d1, d2;

d1 = sqrt((coordinate[**1**].x - coordinate[**3**].x)\*(coordinate[**1**].x - coordinate[**3**].x) + (coordinate[**1**].y - coordinate[**3**].y)\*(coordinate[**2**].y - coordinate[**3**].y));

d2 = sqrt((coordinate[**2**].x - coordinate[**0**].x)\*(coordinate[**2**].x - coordinate[**0**].x) + (coordinate[**2**].y - coordinate[**0**].y)\*(coordinate[**2**].y - coordinate[**0**].y));

**double** ABC = (a \* a + b \* b - d2 \* d2) / (**2** \* a \* b);

**double** BCD = (b \* b + c \* c - d1 \* d1) / (**2** \* b \* c);

**double** CDA = (c \* c + d \* d - d1 \* d1) / (**2** \* c \* d);

**double** DAB = (d \* d + a \* a - d2 \* d2) / (**2** \* d \* a);

**if**(ABC != BCD || ABC != CDA || ABC != DAB || a!=b || a!=c || a!=d) **throw** std::logic\_error("It`s not a square");

//if((coordinate[1].x - coordinate[2].x != coordinate[1].y - coordinate[2].y) || (coordinate[1].x == coordinate[2].x && coordinate[1].y == coordinate[2].y)) throw std::logic\_error("This are incorrect coordinates");

//if(coordinate[1].x - coordinate[2].x != coordinate[1].y - coordinate[2].y) throw std::logic\_error("This is not square");

}

Point Square::center() **const** {

**return** Point((coordinate[**0**].x + coordinate[**2**].x) / **2**, (coordinate[**0**].y + coordinate[**2**].y) / **2**);

}

**double** Square::area() **const** {

//const double dx = coordinate[1].x - coordinate[3].x;

//const double dy = coordinate[1].y - coordinate[3].y;

//return std::abs(dx \* dy);

**return** pow(sqrt((coordinate[**0**].x - coordinate[**3**].x)\*(coordinate[**0**].x - coordinate[**3**].x) + (coordinate[**0**].y - coordinate[**3**].y)\*(coordinate[**0**].y - coordinate[**3**].y)), **2**);

}

**void** Square::print(std::ostream& os) **const** {

std::cout << "Square coordinates:" << std::endl;

os << coordinate[**0**] << std::endl;

os << coordinate[**1**] << std::endl;

os << coordinate[**2**] << std::endl;

os << coordinate[**3**] << std::endl;

}

**main.cpp:**

#include <iostream>

#include <vector>

#include "point.h"

#include "figure.h"

#include "octagon.h"

#include "triangle.h"

#include "square.h"

**void** **help**() {

std::cout << "add = add figure" << std::endl;

std::cout << "delete = delete figure" << std::endl;

std::cout << "print = show information about figure" << std::endl;

std::cout << "print\_all = show information about all figures" << std::endl;

std::cout << "size = the size of our array of figures" << std::endl;

std::cout << "all\_area = the sum area of all figures" << std::endl;

std::cout << "exit = exit" << std::endl;

}

**void** **simple\_add**(std::vector<Figure\*>& figures) {

std::cout << "Press o to add octagon, t to add triangle, s to add square" << std::endl;

std::string info;

Figure\* f = nullptr;

std::cin >> info;

try {

**if**(info == "t") {

std::cout << "Insert 3 coordinates of triangle" << std::endl;

f = **new** Triangle(std::cin);

} **else** **if** (info == "s") {

std::cout << "Insert 4 coordinates of square" << std::endl;

f = **new** Square(std::cin);

} **else** **if** (info == "o") {

std::cout << "Insert 8 coordinates of octagon" << std::endl;

f = **new** Octagon(std::cin);

}

}

**catch** (std::logic\_error& err) {

std::cout << err.what() << std::endl;

}

**if**(f != nullptr) {

figures.push\_back(**dynamic\_cast**<Figure\*>(f));

}

}

/\* std::vector<Figure\*> delete\_el(std::vector<Figure\*>& figures, int del) {

int i = 0;

std::vector<Figure\*> n\_figures;

while(i < figures.size()) {

if(i != del) {

n\_figures.push\_back(figures[i]);

}

i++;

}

figures.clear();

return n\_figures;

} \*/

**int** **main**() {

std::vector<Figure\*> figures;

std::string data;

**int** i;

help();

**while**(std::cin >> data) {

**if**(data == "add") {

simple\_add(figures);

} **else** **if**(data == "delete") {

std::cout << "Index = ";

std::cin >> i;

**if**(i < **0** || i >= figures.size()) {

std::cout << "Incorrect index" << std::endl;

} **else** {

//delete figures[i];

//figures = delete\_el(figures, i);

figures.erase(figures.begin() + i);

}

} **else** **if**(data == "print") {

std::cout << "Index = ";

std::cin >> i;

**if**(i < **0** || i >= figures.size()) {

std::cout << "Incorrect index" << std::endl;

} **else** {

std::cout << "Coordinates:" << std::endl;

std::cout << \*figures[i];

std::cout << "Center:" << std::endl;

std::cout << figures[i]->center();

std::cout << "Area:" << std::endl;

std::cout << figures[i]->area() << std::endl;

}

} **else** **if**(data == "print\_all") {

**for**(**int** j = **0**; j < figures.size(); j++) {

std::cout << j << " figure" << std::endl;

std::cout << \*figures[j] << std::endl;

std::cout << "Center:" << std::endl;

std::cout << figures[j]->center();

std::cout << "Area:" << std::endl;

std::cout << figures[j]->area() << std::endl;

}

} **else** **if**(data == "size") {

std::cout << figures.size() << std::endl;

} **else** **if**(data == "all\_area") {

**double** b;

**for**(**int** j = **0**; j < figures.size(); j++) {

b += figures[j]->area();

}

std::cout << "Area of all figures: " << b << std::endl;

} **else** **if**(data == "exit") {

**for**(**int** j = **0**; j < figures.size(); j++) {

**delete** figures[j];

}

**return** **0**;

}

}

**return** **0**;

}

**2. Ссылка на репозиторий на Github**

[https://github.com/mmaxim2710/oop\_exercise\_0](https://github.com/mmaxim2710/oop_exercise_01)3

**3.Набор testcases**

1)

add

o

4 4 4

size

print

0

all\_area

2)

add

s

0 0 6 6

print\_all

add

t

0 0 0 3 3 0

print\_all

size

delete

0

print

0

exit

**4. Результат выполнения тестов**

1)

1

Coordinates:

Octagon coordinates:

(8, 4)

(6.82843, 6.82843)

(4, 8)

(1.17157, 6.82843)

(0, 4)

(1.17157, 1.17157)

(4, 0)

(6.82843, 1.17157)

Center:

(4, 4)

Area:

45.2548

Area of all figures: 45.2548

2)

0 figure

Square coordinates:

(0, 0)

(0, 6)

(6, 6)

(6, 0)

Center:

(3, 3)

Area:

36

0 figure

Square coordinates:

(0, 0)

(0, 6)

(6, 6)

(6, 0)

Center:

(3, 3)

Area:

36

1 figure

Triangle coordinates

(0, 0)

(0, 3)

(3, 0)

Center:

(1, 1)

Area:

4.5

2

1

Coordinates:

Triangle coordinates

(0, 0)

(0, 3)

(3, 0)

Center:

(1, 1)

Area:

4.5

**5. Объяснение результатов программы**

Данная программа состоит из нескольких частей, описывающих соответствующие классы.

point.h: Описание класса точки. Класс имеет две переменные, соответствующие координатам x и y, конструктор, обнуляющий эти переменные, конструктор, принимающий 2 числа и присваивающий их переменным ох и оу и функции, возвращающие значения х и у. Так же перегружен оператор вывода.  
point.cpp: Реализация класса, описанного в point.h.

figure.h: Описание класса фигуры. Класс имеет виртуальные методы: вычисление геометрического центра фигуры, вычисление площади, функция вывода, которая используется в перегрузке оператора вывода для того, чтобы для каждой фигуры не перегружать оператор.

figure.cpp: Связь функции вывода с оператором <<.

octagon.h: Описание класса 8-ми угольника — потомка класса figure. Принимает координаты центра, радиус описанной окружности (или расстояние от центра до любой вершины) и по формуле строит координаты всех вершин. Гарантированно данная фигура является правильным 8-ми угольником. Перегружены методы центра, площади, вывода.

octagon.cpp: реализация octagon.h.

triangle.h: Описание класса триугольника — потомка класса figure. Принимает координаты 3 точек, проверяет, чтобы сумма двух любых сторон была больше третей.

Перегружены методы центра, площади, вывода.

triangle.cpp: реализация triangle.h.

square.h: Описание класса квадрата — потомка класса figure. Принимает координаты 2х противоположных точек, проверяет, что разность х и у каждой точки равна.

Перегружены методы центра, площади, вывода.

square.cpp – реализация square.h.

main.cpp: Реализованы 3 функции: функция вывода меню, функция добавления фигуры, функция, помогающая удалять фигуру. Функция добавления запрашивает необходимые данные (2 точки, 3 точки, точка и радиус), создает соответствующие точки, по точкам создает объект соответсвующей фигуры и передает его вектору. Функция удаления принимает старый вектор, откуда удалили элемент, создает новый элемент и перекидывает объекты из старого в новый за исключением пустого места.

Вывод: Проделав данную работу я научился работать с наследованием классов, выявил плюсы наследования: если для решения задачи необходимо написать несколько классов, которые имеют одинаковые свойства, то можно написать один класс, от которого будут наследоваться данные классы.