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 5**

**Наследование, полиморфизм**

|  |  |
| --- | --- |
| Студент: | Валов Вадим Вячеславович |
| Группа: | М8О-208Б-18 |
| Преподаватель: | Журалвев А.А. |
|  |  |
| Оценка: |  |
| Дата: |  |

# 1. Код программы на языке С++

**1. Файл modulo.h**

#pragma once

#include <iostream>

#include <vector>

**struct** Point {

**double** x, y;

};

**double** **calculateDistance**(**const** Point& lhs, **const** Point& rhs);

**bool** **operator**<(**const** Point& lhs, **const** Point& rhs);

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

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

std::ostream& **operator**<<(std::ostream& os, **const** std::vector<Point> v);

**class** **Figure** {

**public:**

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

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

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

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

};

**class** **Rhombus** : **public** Figure {

**public:**

Rhombus(**const** Point& p1, **const** Point& p2, **const** Point& p3, **const** Point& p4);

Point Center() **const** override;

**double** Square() **const** override;

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

**private:**

std::vector<Point> points;

**double** smallerDiagonal, biggerDiagonal;

};

**class** **Pentagon** : **public** Figure {

**public:**

Pentagon(**const** Point& p1, **const** Point& p2, **const** Point& p3, **const** Point& p4, **const** Point& p5);

Point Center() **const** override;

**double** Square() **const** override;

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

**private:**

std::vector<Point> points;

};

**class** **Hexagon** : **public** Figure {

**public:**

Hexagon(**const** Point& p1, **const** Point& p2, **const** Point& p3, **const** Point& p4, **const** Point& p5, **const** Point& p6);

Point Center() **const** override;

**double** Square() **const** override;

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

**private:**

std::vector<Point> points;

};

**2. Файл modulo.cpp**

#include "Figures.hpp"

#include <cmath>

#include <algorithm>

#include <iomanip>

**double** **calculateDistance**(**const** Point& lhs, **const** Point& rhs) {

**return** sqrt(pow(rhs.x - lhs.x, **2**) + pow(rhs.y - lhs.y, **2**));

}

**bool** **operator**<(**const** Point& lhs, **const** Point& rhs) {

**if**(lhs.x != rhs.x) {

**return** lhs.x < rhs.x;

}

**return** lhs.y < rhs.y;

}

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 << std::fixed << std::setprecision(**3**) << "[" << p.x << ", " << p.y << "]";

**return** os;

}

std::ostream& **operator**<<(std::ostream& os, **const** std::vector

<Point> v) {

**for**(**unsigned** i = **0**; i < v.size(); ++i) {

os << v[i];

**if**(i != v.size() - **1**) {

os << ", ";

}

}

**return** os;

}

**double** checkIfRhombus(**const** Point& p1, **const** Point& p2, **const**

Point& p3, **const** Point& p4) {

**double** d1 = calculateDistance(p1, p2);

**double** d2 = calculateDistance(p1, p3);

**double** d3 = calculateDistance(p1, p4);

**if**(d1 == d2) {

**return** d3;

} **else** **if**(d1 == d3) {

**return** d2;

} **else** **if**(d2 == d3) {

**return** d1;

} **else** {

**throw** std::invalid\_argument("Entered coordinates are not forming Rhombus. Try entering new coordinates");

}

}

Rhombus::Rhombus(**const** Point& p1, **const** Point& p2, **const** Point& p3, **const** Point& p4) {

try {

**double** d1 = checkIfRhombus(p1, p2, p3, p4);

**double** d2 = checkIfRhombus(p2, p1, p3, p4);

**double** d3 = checkIfRhombus(p3, p1, p2, p4);

**double** d4 = checkIfRhombus(p4, p1, p2, p3);

**if**(d1 == d2 || d1 == d4) {

**if**(d1 < d3) {

smallerDiagonal = d1;

biggerDiagonal = d3;

} **else** {

smallerDiagonal = d3;

biggerDiagonal = d1;

}

} **else** **if**(d1 == d3) {

**if**(d1 < d2) {

smallerDiagonal = d1;

biggerDiagonal = d2;

} **else** {

smallerDiagonal = d2;

biggerDiagonal = d1;

}

}

} **catch**(std::exception& e) {

**throw** std::invalid\_argument(e.what());

**return**;

}

points.push\_back(p1);

points.push\_back(p2);

points.push\_back(p3);

points.push\_back(p4);

}

Point Rhombus::Center() **const** {

**if**(calculateDistance(points[**0**], points[**1**]) == smallerDiagonal ||

calculateDistance(points[**0**], points[**1**]) == biggerDiagonal) {

**return** {((points[**0**].x + points[**1**].x) / **2.0**), ((points[**0**].y + points[**1**].y) / **2.0**)};

} **else** **if**(calculateDistance(points[**0**], points[**2**]) == smallerDiagonal ||

calculateDistance(points[**0**], points[**2**]) == biggerDiagonal) {

**return** {((points[**0**].x + points[**2**].x) / **2.0**), ((points[**0**].y + points[**2**].y) / **2.0**)};

} **else** {

**return** {((points[**0**].x + points[**3**].x) / **2.0**), ((points[**0**].y + points[**3**].y) / **2.0**)};

}

}

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

**return** smallerDiagonal \* biggerDiagonal / **2.0**;

}

**void** Rhombus::Print(std::ostream& os) **const** {

**if**(points.size()) {

os << "Rhombus: " << points << std::endl;

}

}

Pentagon::Pentagon(**const** Point& p1, **const** Point& p2, **const** Point& p3,

**const** Point& p4, **const** Point& p5) {

points.push\_back(p1);

points.push\_back(p2);

points.push\_back(p3);

points.push\_back(p4);

points.push\_back(p5);

}

**double** triangleSquare(**const** Point& p1, **const** Point& p2, **const** Point& p3) {

**return** **0.5** \* fabs((p1.x - p3.x) \* (p2.y - p3.y) - (p2.x - p3.x) \* (p1.y - p3.y));

}

Point Pentagon::Center() **const** {

Point insideFigure{**0**, **0**};

Point result{**0**, **0**};

**double** square = **this**->Square();

**for**(**unsigned** i = **0**; i < points.size(); ++i) {

insideFigure.x += points[i].x;

insideFigure.y += points[i].y;

}

insideFigure.x /= points.size();

insideFigure.y /= points.size();

**for**(**unsigned** i = **0**; i < points.size(); ++i) {

**double** tempSquare = triangleSquare(points[i], points[(i + **1**) % points.size()],

insideFigure);

result.x += tempSquare \* (points[i].x + points[(i + **1**) % points.size()].x

+ insideFigure.x) / **3.0**;

result.y += tempSquare \* (points[i].y + points[(i + **1**) % points.size()].y

+ insideFigure.y) / **3.0**;

}

result.x /= square;

result.y /= square;

**return** result;

}

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

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

**for**(**unsigned** i = **0**; i < points.size(); ++i) {

Point p1 = i ? points[i - **1**] : points[points.size() - **1**];

Point p2 = points[i];

result += (p1.x - p2.x) \* (p1.y + p2.y);

}

**return** fabs(result) / **2.0**;

}

**void** Pentagon::Print(std::ostream& os) **const** {

os << "Pentagon: " << points << std::endl;

}

Hexagon::Hexagon(**const** Point& p1, **const** Point& p2, **const** Point& p3,

**const** Point& p4, **const** Point& p5, **const** Point& p6) {

points.push\_back(p1);

points.push\_back(p2);

points.push\_back(p3);

points.push\_back(p4);

points.push\_back(p5);

points.push\_back(p6);

}

Point Hexagon::Center() **const** {

Point insideFigure{**0**, **0**};

Point result{**0**, **0**};

**double** square = **this**->Square();

**for**(**unsigned** i = **0**; i < points.size(); ++i) {

insideFigure.x += points[i].x;

insideFigure.y += points[i].y;

}

insideFigure.x /= points.size();

insideFigure.y /= points.size();

**for**(**unsigned** i = **0**; i < points.size(); ++i) {

**double** tempSquare = triangleSquare(points[i], points[(i + **1**) % points.size()],

insideFigure);

result.x += tempSquare \* (points[i].x + points[(i + **1**) % points.size()].x

+ insideFigure.x) / **3.0**;

result.y += tempSquare \* (points[i].y + points[(i + **1**) % points.size()].y

+ insideFigure.y) / **3.0**;

}

result.x /= square;

result.y /= square;

**return** result;

}

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

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

**for**(**unsigned** i = **0**; i < points.size(); ++i) {

Point p1 = i ? points[i - **1**] : points[points.size() - **1**];

Point p2 = points[i];

result += (p1.x - p2.x) \* (p1.y + p2.y);

}

**return** fabs(result) / **2.0**;

}

**void** Hexagon::Print(std::ostream& os) **const** {

os << "Hexagon: " << points << std::endl;

}

**3. Файл main.cpp**

#include <iostream>

#include <vector>

#include "point.hpp"

#include "figure.hpp"

#include "rhombus.hpp"

#include "pentagon.hpp"

#include "hexagon.hpp"

**int** **get\_command**() {

**int** command;

std::cin >> command;

**return** command;

}

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

**int** command1, command2;

std::vector<Figure\*> figures;

std::cout << "1 - add figure to the vector**\n**"

"2 - delete figure from the vector**\n**"

"3 - call common functions for the whole vector**\n**"

"4 - get total area of figures in vector**\n**"

"0 - exit**\n**";

**while**((command1 = get\_command()) != **0**) {

**if**(command1 == **1**) {

Figure\* f;

std::cout << "1 - Rhombus, 2 - Pentagon, 3 - Hexagon" << std::endl;

std::cin >> command2;

try {

**if**(command2 == **1**) {

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

} **else** **if**(command2 == **2**) {

f = **new** Pentagon{std::cin};

} **else** **if**(command2 == **3**) {

f = **new** Hexagon{std::cin};

} **else** {

std::cout << "Wrong input" << std::endl;

}

figures.push\_back(f);

} **catch**(std::exception& e) {

std::cerr << e.what() << std::endl;

}

} **else** **if**(command1 == **2**) {

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

std::cin >> command2;

**if**(command2 < **static\_cast**<**int**>(figures.size())) {

**delete** figures[command2];

figures.erase(figures.begin() + command2);

} **else** {

std::cout << "Element with such index doesn't exist" << std::endl;

}

} **else** **if**(command1 == **3**) {

**for**(**const** **auto**& figure : figures) {

figure->Print(std::cout);

std::cout << figure->Center() << std::endl;

std::cout << figure->Square() << std::endl;

}

} **else** **if**(command1 == **4**) {

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

**for**(**const** **auto**& figure : figures) {

result += figure->Square();

}

std::cout << result << std::endl;

} **else** **if**(command1 == **5**) {

**break**;

} **else** {

std::cout << "Wrong command" << std::endl;

}

}

**for**(**const** **auto**& f : figures) {

**delete** f;

}

**return** **0**;

}

**2. Ссылка на репозиторий на Github**

https://github.com/vindosVP/oop\_exercise\_03

**3.Набор тестов**

3

**4. Результат выполнения тестов**

**Объяснение работы программы**

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.