**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 3**

Тема: Наследования, полиформизм.

Студент: Егорова Анна (староста)

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

1. Вычисление геометрического центра фигуры;

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;

3. Вычисление площади фигуры;

Создать программу, которая позволяет:

• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.

• Сохранять созданные фигуры в динамический массив std::vector<Figure\*>

• Вызывать для всего массива общие функции (1-3 см. выше).Т.е. распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь.

• Необходимо уметь вычислять общую площадь фигур в массиве.

• Удалять из массива фигуру по индексу;

Фигуры: восьмиугольник, шестиугольник, пятиугольник.

1. Описание программы

Программа состоит из главного файла main.cpp, CmakeLists.txt – утилиты, которая собирает все файлы и папки source, в которой находятся все библиотеки и файлы, связанные с построением и вычислением всех величин фигур из моего варианта.

В ООП существует 3 основных принципа построения классов:

1. Инкапсуляция – это свойство, позволяющее объединить в классе и данные и методы, работающие с ним и скрыть детали реализации от пользователя.

2. Наследование – Это свойство, позволяющее создавать новый класс-потомок на основе уже существующего, при этом все характеристики класса-родителя присваиваются классу-потомку.

3. Полиформизм – свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

1. Набор testcases

Тест 1:

Пятиугольник: 2 2 4

Шестиугольник: 3 4 5

Восьмиугольник: 4 5 4

Тест 2:

Пятиугольник: 5 5 7

Шестиугольник: 7 6 5

Восьмиугольник: 3 10 4

1. Результаты выполнения тестов.

Тест 1:

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3$ ./main

add = add figure

delete = delete figure

print = show information about figure

print\_all = show information about all figures

size = the size of our array of figures

all\_area = the sum area of all figures

exit = exit

add

Press pe to add pentagon, o to add octagon, h to add Hexagon

pe

Please, your center coordinate and circle radius

3 4 5

print pe

Index : Coordinates of Pentagon:

Point : (8.000,4.000)

Point : (4.545,8.755)

Point : (-1.045,6.939)

Point : (-1.045,1.061)

Point : (4.545,-0.755)

center Point : (3.000,4.000)

Area : 118.882

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3$ ./main

add = add figure

delete = delete figure

print = show information about figure

print\_all = show information about all figures

size = the size of our array of figures

all\_area = the sum area of all figures

exit = exit

add

Press pe to add pentagon, o to add octagon, h to add Hexagon

pe

Please, your center coordinate and circle radius

2 2 4

add

Press pe to add pentagon, o to add octagon, h to add Hexagon

h

Please, your center coordinate and circle radius

3 4 5

add

Press pe to add pentagon, o to add octagon, h to add Hexagon

o

Please, your center coordinate and circle radius

4 5 4

print\_all

0-st figure

Coordinates of Pentagon:

Point : (6.000,2.000)

Point : (3.236,5.804)

Point : (-1.236,4.351)

Point : (-1.236,-0.351)

Point : (3.236,-1.804)

center Point : (2.000,2.000)

Area : 76.085

1-st figure

Coordinates of Hexagon:

Point : (8.000,4.000)

Point : (5.500,8.330)

Point : (0.500,8.330)

Point : (-2.000,4.000)

Point : (0.500,-0.330)

Point : (5.500,-0.330)

center Point : (3.000,4.000)

Area : 64.952

2-st figure

Coordinates of octagon:

Point : (8.000,5.000)

Point : (6.828,7.828)

Point : (4.000,9.000)

Point : (1.172,7.828)

Point : (0.000,5.000)

Point : (1.172,2.172)

Point : (4.000,1.000)

Point : (6.828,2.172)

center Point : (4.000,5.000)

Area : 45.255

size

3

all\_area

Area of all fiqures : 186.291

delete

Index = 2

print\_all

0-st figure

Coordinates of Pentagon:

Point : (6.000,2.000)

Point : (3.236,5.804)

Point : (-1.236,4.351)

Point : (-1.236,-0.351)

Point : (3.236,-1.804)

center Point : (2.000,2.000)

Area : 76.085

1-st figure

Coordinates of Hexagon:

Point : (8.000,4.000)

Point : (5.500,8.330)

Point : (0.500,8.330)

Point : (-2.000,4.000)

Point : (0.500,-0.330)

Point : (5.500,-0.330)

center Point : (3.000,4.000)

Area : 64.952

Тест 2:

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3$ ./main

add = add figure

delete = delete figure

print = show information about figure

print\_all = show information about all figures

size = the size of our array of figures

all\_area = the sum area of all figures

exit = exit

add

Press pe to add pentagon, o to add octagon, h to add Hexagon

pe

Please, your center coordinate and circle radius

5 5 7

add

Press pe to add pentagon, o to add octagon, h to add Hexagon

h

Please, your center coordinate and circle radius

7 6 5

add

Press pe to add pentagon, o to add octagon, h to add Hexagon

o

Please, your center coordinate and circle radius

3 10 4

print\_all

0-st figure

Coordinates of Pentagon:

Point : (12.000,5.000)

Point : (7.163,11.657)

Point : (-0.663,9.114)

Point : (-0.663,0.886)

Point : (7.163,-1.657)

center Point : (5.000,5.000)

Area : 233.009

1-st figure

Coordinates of Hexagon:

Point : (12.000,6.000)

Point : (9.500,10.330)

Point : (4.500,10.330)

Point : (2.000,6.000)

Point : (4.500,1.670)

Point : (9.500,1.670)

center Point : (7.000,6.000)

Area : 64.952

2-st figure

Coordinates of octagon:

Point : (7.000,10.000)

Point : (5.828,12.828)

Point : (3.000,14.000)

Point : (0.172,12.828)

Point : (-1.000,10.000)

Point : (0.172,7.172)

Point : (3.000,6.000)

Point : (5.828,7.172)

center Point : (3.000,10.000)

Area : 45.255

1. Листинг программы

**MAIN.CPP**

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3$ cat main.cpp

/\* Егорова Анна Владимировна М8O-207Б-18

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

1. Вычисление геометрического центра фигуры;

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры; 3. Вычисление площади фигуры;

Фигуры: восьмиугольник, шестиугольник, пятиугольник.

Создать программу, которая позволяет:

• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.

• Сохранять созданные фигуры в динамический массив std::vector<Figure\*>

• Вызывать для всего массива общие функции (1-3 см. выше).Т.е. распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь.

• Необходимо уметь вычислять общую площадь фигур в массиве.

• Удалять из массива фигуру по индексу;

\*/

#include "source/Point\_figure.hpp"

#include "source/Octagon.hpp"

#include "source/Hexagon.hpp"

#include "source/Pentagon.hpp"

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 pe to add pentagon, o to add octagon, h to add Hexagon" << std::endl;

std::string inf;

Pentagon \*pe = nullptr;

Hexagon \*h = nullptr;

Octagon \*o = nullptr;

std::cin >> inf;

if (inf == "pe") {

std::cout << "Please, your center coordinate and circle radius" << std::endl;

double x1,y1,R;

std::cin >> x1 >> y1 >> R;

Point p(x1,y1);

try {

pe = new Pentagon(p,R);

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

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

delete pe;

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

}

} else if (inf == "h") {

std::cout << "Please, your center coordinate and circle radius" << std::endl;

double x1,y1,R;

std::cin >> x1 >> y1 >> R;

Point p(x1,y1);

try {

h = new Hexagon(p,R);

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

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

delete h;

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

}

} else if (inf == "o") {

std::cout << "Please, your center coordinate and circle radius" << std::endl;

double x1,y1,R;

std::cin >> x1 >> y1 >> R;

Point p(x1,y1);

try {

o = new Octagon(p,R);

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

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

delete o;

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

}

}

}

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::cout << std::fixed;

std::cout.precision(3);

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 << "Error" << std::endl;

} else {

delete figures[i];

figures = delete\_el(figures, i);

}

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

std::cout << "Index : ";

std::cin >> i;

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

std::cout << "Error" << std::endl;

} else {

figures[i]->output();

std::cout << "center ";

figures[i]->get\_centre().P\_output();

std::cout << "Area : " << figures[i]->get\_area() << std::endl;

}

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

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

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

figures[j]->output();

std::cout << "center ";

figures[j]->get\_centre().P\_output();

std::cout << "Area : " << figures[j]->get\_area() << std::endl;

}

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

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

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

double b = 0;

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

b = b + figures[j]->get\_area();

}

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

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

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

delete figures[j];

}

return 0;

}

}

return 0;

}

**CmakeLists.txt**

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3$ cat CMakeLists.txt

cmake\_minimum\_required(VERSION 2.8)

project(oop)

add\_executable(main main.cpp source/point\_figure.cpp source/octagon.cpp source/hexagon.cpp source/pentagon.cpp)

**Source**

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3$ cd source

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat hexagon.cpp

#include "Hexagon.hpp"

Hexagon::Hexagon() : z(0,0), R(0)

{}

Hexagon::Hexagon(Point x, double Rad) : z(x), R(Rad)

{

if(R <= 0) {

throw std::logic\_error("always R > 0");

}

}

void Hexagon::output() {

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

double angle;

for(int i = 0; i < 6; i++) {

angle = M\_PI \* i / 3;

Point(R \* cos(angle) + z.x(),R \* sin(angle) + z.y()).P\_output() ;

}

}

double Hexagon:: get\_area() {

return 3 \* sin(M\_PI / 3) \* R \* R;

}

Point Hexagon::get\_centre() {

return z;

}

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat Hexagon.hpp

#ifndef HEXAGON\_HPP

#define HEXAGON\_HPP

#include "Point\_figure.hpp"

class Hexagon : public Figure {

private:

Point z;

double R;

public:

Hexagon();

Hexagon(Point x, double Rad);

void output() override;

double get\_area() override;

Point get\_centre() override;

};

#endif

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat Octagon.hpp

#ifndef OCTAGON\_HPP

#define OCTAGON\_HPP

#include "Point\_figure.hpp"

class Octagon : public Figure {

private:

Point z;

double R;

public:

Octagon();

Octagon(Point x, double Rad);

void output() override;

double get\_area() override;

Point get\_centre() override;

};

#endif

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat octagon.cpp

#include "Octagon.hpp"

Octagon::Octagon() : z(0,0), R(0)

{}

Octagon::Octagon(Point x, double Rad) : z(x), R(Rad)

{

if(R <= 0) {

throw std::logic\_error("always R > 0");

}

}

void Octagon::output() {

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

double angle;

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

angle = M\_PI \* i / 4;

Point(R \* cos(angle) + z.x(),R \* sin(angle) + z.y()).P\_output() ;

}

}

double Octagon:: get\_area() {

return 4 \* sin(M\_PI / 4) \* R \* R;

}

Point Octagon::get\_centre() {

return z;

}

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat pentagon.cpp

#include "Pentagon.hpp"

Pentagon::Pentagon() : z(0,0), R(0)

{}

Pentagon::Pentagon(Point x, double Rad) : z(x), R(Rad)

{

if(R <= 0) {

throw std::logic\_error("always R > 0");

}

}

void Pentagon::output() {

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

double angle;

for(int i = 0; i < 5; i++) {

angle = 2\*M\_PI \* i / 5;

Point(R \* cos(angle) + z.x(), R \* sin(angle) + z.y()).P\_output() ;

}

}

double Pentagon:: get\_area() {

return 5 \* sin(2\*M\_PI / 5) \* R \* R;

}

Point Pentagon::get\_centre() {

return z;

}

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat Pentagon.hpp

#ifndef PENTAGON\_HPP

#define PENTAGON\_HPP

#include "Point\_figure.hpp"

class Pentagon : public Figure {

private:

Point z;

double R;

public:

Pentagon();

Pentagon(Point x, double Rad);

void output() override;

double get\_area() override;

Point get\_centre() override;

};

#endif

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat point\_figure.cpp

#include "Point\_figure.hpp"

Point::Point() : ox(0), oy(0)

{}

Point::Point(double a, double b): ox(a), oy(b)

{}

double Point::x() {

return ox;

}

double Point::y() {

return oy;

}

void Point::P\_output() {

std::cout << "Point : " << "(" << x() << "," << y() << ")" << std::endl;

}

anyaegorovaa@LAPTOP-9PIFT2VJ:~/2kurs/oop/l3/source$ cat Point\_figure.hpp

#ifndef POINT\_FIGURE\_HPP

#define POINT\_FIGURE\_HPP

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <vector>

class Point {

private:

double ox,oy;

public:

Point();

Point(double a, double b);

double x();

double y();

void P\_output();

};

class Figure {

public:

virtual void output() = 0;

virtual double get\_area() = 0;

virtual Point get\_centre() = 0;

};

#endif

1. Вывод

В данной работе я познакомилась с новой для себя, объектно-ориентированной парадигмой программирования. В отличие от знакомой мне процедурной парадигмы, где атрибуты и поведения обычно разделяются, при объектно-ориентированном проектировании атрибуты и поведения размещаются в рамках одного объекта. Конечно, они не являются взаимоисключающими, и многие программы, написанные на

С++, принадлежат обеим парадигмам. Я приобрела навыки проектирования классов и работы с ними. Фундаментальные концепции ООП полиморфизм интерфейса (печать параметров фигуры и подсчет площади) и реализации (параметры фигуры), наследование в виде производных классов Trapeze, Rhomb и Рentagon oт класса Figure, полиморфизм в виде переопределении методов Print и Square. Удобно использовать объекты, не задумываясь о внутренней реализации. инкапсуляция, наследование и также отражены в моей работе.

1. Список литературы.

1. Полиформизм в С++ [Электронный ресурс]. URL: https://habr.com/ru/post/37576/

2. Наследование в С++ [Электронный ресурс]. URL: https://habr.com/ru/post/445948/