**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 3**

Тема: Механизмы наследования в С++

Студент: Ильин Илья Олегович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

# Оглавление

1. [Оглавление 2](#_Toc53741091)
2. [Постановка задачи 3](#_Toc53741092)
3. [Описание программы 4](#_Toc53741093)
4. [Набор тестов 5](#_Toc53741094)

[test\_01.txt 5](#_Toc53741095)

[test\_02.txt 5](#_Toc53741096)

[test\_03.txt 6](#_Toc53741097)

1. [Листинг программы 8](#_Toc53741098)
2. [Список литературы 14](#_Toc53741099)

# 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;

Вариант 7.

Фигуры: треугольник, 6-угольник, 8-угольник.

# Описание программы

Сначала, создадим структуру *Point,* которая хранит координаты точки на плоскости. Для неё перегрузим методы ввода и вывода.

В файле *figure.cpp* реализуем базовый класс *Figure*, содержащий защищённые поля *\_center, \_circumradius, \_vertices*, которые представляю собой координаты центра фигуры, радиус описанной окружности, вектор вершин. Также, класс содержит конструктор, деструктор и виртуальные методы подсчёта площади, вершин и получения вектора вершин.

Также, были реализованы классы *Triangle, Hexagon, Octagon* с перегрузками раннее указанных методов. Конструкторы этих классов используют конструктор базового класса.

В файле *main.cpp* реализована интерактивная часть программы. На ввод подаются команды, затем либо выполняются соответствующие действия, либо выводятся сообщения об ошибке.

# Набор тестов

Поскольку программа является интерактивной, тесты содержат несколько запросов, после которых с помощью команд *print* и *area* происходит вывод результата.

## test\_01.txt

Проверка на корректность подсчитывания площади фигур и общей площади.

add triangle 0 10 10

print

**>>> Type: triangle**

**Center: (0,10)**

**Area: 129.904**

**Vertices:**

**(10,10)**

**(9.77569,12.1062)**

**(9.11281,14.1179)**

add hexagon -1 -1 20

print

**>>> Type: triangle**

**Center: (0,10)**

**Area: 129.904**

**Vertices:**

**(10,10)**

**(9.77569,12.1062)**

**(9.11281,14.1179)**

**>>> Type: hexagon**

**Center: (-1,-1)**

**Area: 1039.23**

**Vertices:**

**(19,-1)**

**(18.8875,1.11809)**

**(18.5514,3.21235)**

**(17.9953,5.25924)**

**(17.2256,7.23572)**

**(16.2509,9.11958)**

area

**Summary area is 1169.13**

## test\_02.txt

Проверка корректности обработки исключений. Рассматриваются всевозможные комбинации неправильных запросов. При этом, проверяется, чтобы не были добавлены неправильные обьекты.

adf triangle 0 1 2

**Invalid command**

add haxd 3 2 1

**Invalid object type**

add hexagon 0 1 2

add hexagon a 2 3

**stod: no conversion**

**Invalid command**

add triangle a a 3

**stod: no conversion**

**Invalid command**

add octagon 1 a 3

**stod: no conversion**

**Invalid command**

add hexagon 1 2 a

**stod: no conversion**

add triangle 1 2 -10

**Радиус не может быть <= 0**

print

**>>> Type: hexagon**

**Center: (0,1)**

**Area: 10.3923**

**Vertices:**

**(2,1)**

**(1.98875,1.21181)**

**(1.95514,1.42124)**

**(1.89953,1.62592)**

**(1.82256,1.82357)**

**(1.72509,2.01196)**

## test\_03.txt

Проверка функции удаления.

add hexagon 6 6 6

add octagon 8 8 8

print

**>>> Type: hexagon**

**Center: (6,6)**

**Area: 93.5307**

**Vertices:**

**(12,6)**

**(11.9663,6.63543)**

**(11.8654,7.26371)**

**(11.6986,7.87777)**

**(11.4677,8.47072)**

**(11.1753,9.03587)**

**>>> Type: octagon**

**Center: (8,8)**

**Area: 362.039**

**Vertices:**

**(16,8)**

**(15.9747,8.63595)**

**(15.8989,9.26787)**

**(15.7731,9.89177)**

**(15.5981,10.5037)**

**(15.3751,11.0998)**

**(15.1053,11.6762)**

**(14.7906,12.2294)**

delete 1

print

**>>> Type: octagon**

**Center: (8,8)**

**Area: 362.039**

**Vertices:**

**(16,8)**

**(15.9747,8.63595)**

**(15.8989,9.26787)**

**(15.7731,9.89177)**

**(15.5981,10.5037)**

**(15.3751,11.0998)**

**(15.1053,11.6762)**

**(14.7906,12.2294)**

delete 1

**Invalid index**

print

**>>> Type: octagon**

**Center: (8,8)**

**Area: 362.039**

**Vertices:**

**(16,8)**

**(15.9747,8.63595)**

**(15.8989,9.26787)**

**(15.7731,9.89177)**

**(15.5981,10.5037)**

**(15.3751,11.0998)**

**(15.1053,11.6762)**

**(14.7906,12.2294)**

# Листинг программы

//

// main.cpp

// oop\_exercise\_03

//

// Created by Илья Ильин on 16.10.2020.

//

/\*

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;

Вариант 7:

6-угольник, 8-угольник, треугольник

\*/

#include "figure.cpp"

void print\_info() {

std::cout << "+----------------------------+--------------------------------+" << std::endl;

std::cout << "|add [type] [center] [radius]| Добавляет фигуру |" << std::endl;

std::cout << "+----------------------------+--------------------------------+" << std::endl;

std::cout << "| delete [index] | Удаляет фигуру по индексу | " << std::endl;

std::cout << "+----------------------------+--------------------------------+" << std::endl;

std::cout << "| print | Печатает информацию по фигурам | " << std::endl;

std::cout << "+----------------------------+--------------------------------+" << std::endl;

std::cout << "| area | Печатает общую площадь | " << std::endl;

std::cout << "+----------------------------+--------------------------------+" << std::endl;

std::cout << "| types: triangle, hexagon, octagon | " << std::endl;

std::cout << "+-------------------------------------------------------------+" << std::endl;

std::cout << "| example: add triangle 0 10 10 | " << std::endl;

std::cout << "+-------------------------------------------------------------+" << std::endl;

}

int main(int argc, const char \* argv[])

{

print\_info();

std::string cmd = "";

std::vector<Figure\*> figures;

while (std::cin >> cmd) {

try {

if (cmd == "add") {

std::string obj\_type = "";

std::cin >> obj\_type;

Point center;

std::cin >> center;

std::string str\_radius = "";

std::cin >> str\_radius;

double radius = std::stod(str\_radius);

if (obj\_type == "triangle") {

Triangle\* t\_ptr = new Triangle(center, radius);

figures.push\_back(t\_ptr);

}

else if (obj\_type == "hexagon") {

Hexagon\* h\_ptr = new Hexagon(center, radius);

figures.push\_back(h\_ptr);

}

else if (obj\_type == "octagon") {

Octagon\* o\_ptr = new Octagon(center, radius);

figures.push\_back(o\_ptr);

} else {

throw std::invalid\_argument("Invalid object type");

}

}

else if (cmd == "delete") {

size\_t idx = -1;

std::cin >> idx;

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

throw std::out\_of\_range("Invalid index");

}

figures.erase(figures.begin() + (idx - 1));

}

else if (cmd == "print") {

for (auto figure : figures) {

std::vector<Point> vertices;

figure->get\_vertices(vertices);

size\_t n = vertices.size();

std::string obj\_type = "";

switch (n) {

case 3:

obj\_type = "triangle";

break;

case 6:

obj\_type = "hexagon";

break;

case 8:

obj\_type = "octagon";

break;

default:

obj\_type = "unknown";

break;

}

std::cout << ">>> Type: " << obj\_type << std::endl;

std::cout << "Center: " << figure->get\_center() << std::endl;

std::cout << "Area: " << figure->get\_area() << std::endl;

std::cout << "Vertices: " << std::endl;

for (auto vertex : vertices){

std::cout << vertex << std::endl;

}

}

} else if (cmd == "area") {

double summary = 0;

for (auto figure : figures) {

summary += figure->get\_area();

}

std::cout << "Summary area is " << summary << std::endl;

} else {

char c;

while((c =getchar() != '\n')) { }

throw std::invalid\_argument("Invalid command");

}

}

catch (std::exception& ex) {

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

}

}

return 0;

}

//

// figure.cpp

// oop\_exercise\_03

//

// Created by Илья Ильин on 16.10.2020.

//

#include <vector>

#include <iostream>

#include <stdio.h>

#define \_USE\_MATH\_DEFINES

#include <cmath>

struct Point {// Структура для хранения координат точек на плоскости

double X = 0;

double Y = 0;

};

inline std::ostream& operator<<(std::ostream& output, const Point& point {

output << "(" << point.X << "," << point.Y << ")";

return output;

}

inline std::istream& operator>>(std::istream& input, Point& point {

std::string a, b;

input >> a >> b;

point.X = std::stod(a);

point.Y = std::stod(b);

return input;

}

class Figure{

protected:

Point \_center; // Координаты центра многоугольника

double \_circumradius; // Радиус описанной вокруг многоугольника окр-ти

std::vector<Point> \_vertices;

public:

Figure(const Point& center, const double& circumradius) : \_center(center) {

if (circumradius <= 0) {

throw std::invalid\_argument("Радиус не может быть <= 0");

}

\_circumradius = circumradius;

//std::cout << "<<<>>> It's Figure constuctor" << std::endl;

}

~Figure() {

\_center.X = 0;

\_center.Y = 0;

\_circumradius = 0;

\_vertices.clear();

//std::cout << "<<<>>> It's Figure destructor" << std::endl;

}

Point get\_center() const {

return \_center;

}

double get\_circumradius() const {

return \_circumradius;

}

void set\_circumradius(const double& radius) {

if (radius <= 0) {

throw std::invalid\_argument("Радиус не может быть <= 0");

}

\_circumradius = radius;

recount\_vertices();

}

virtual double get\_area() const = 0; // Метод получения площади фигуры

virtual void recount\_vertices() {} // Метод, подсчитывающий координаты вершин фигуры

virtual void get\_vertices(std::vector<Point> &vertices) const {} // Метод получения вектора координат вершин

};

class Triangle : public Figure {

public:

Triangle(const Point& center, const double& circumradius) : Figure(center, circumradius) {

recount\_vertices();

}

double get\_area() const override {

return (3 \* \_circumradius \* \_circumradius \* sqrt(3)) / 4;

}

void recount\_vertices() override {

\_vertices.clear();

Point vertex;

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

vertex.X = \_center.X + \_circumradius \* cos( (M\_2\_PI \* i) / 3 );

vertex.Y = \_center.Y + \_circumradius \* sin( (M\_2\_PI \* i) / 3 );

\_vertices.push\_back(vertex);

}

}

void get\_vertices(std::vector<Point> &vertices) const override {

vertices = \_vertices;

}

};

class Octagon : public Figure {

public:

Octagon(const Point& center, const double& circumradius) : Figure(center, circumradius) {

recount\_vertices();

}

void recount\_vertices() override {

\_vertices.clear();

Point vertex;

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

vertex.X = \_center.X + \_circumradius \* cos( (M\_2\_PI \* i) / 8 );

vertex.Y = \_center.Y + \_circumradius \* sin( (M\_2\_PI \* i) / 8 );

\_vertices.push\_back(vertex);

}

}

void get\_vertices(std::vector<Point> &vertices) const override {

vertices = \_vertices;

}

double get\_area() const override{

return \_circumradius \* \_circumradius \* ( 4 \* sqrt(2) );

}

};

class Hexagon : public Figure {

public:

Hexagon(const Point& center, const double& circumradius) : Figure(center, circumradius) {

recount\_vertices();

}

void recount\_vertices() override {

\_vertices.clear();

Point vertex;

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

vertex.X = \_center.X + \_circumradius \* cos( (M\_2\_PI \* i) / 6 );

vertex.Y = \_center.Y + \_circumradius \* sin( (M\_2\_PI \* i) / 6 );

\_vertices.push\_back(vertex);

}

}

void get\_vertices(std::vector<Point> &vertices) const override {

vertices = \_vertices;

}

double get\_area() const override {

return \_circumradius \* \_circumradius \* ( ( 3 \* sqrt(3) ) / 2 );

}

};

# Список литературы

[1] *Информация по методам и функциям языка C++*.

URL: <http://www.cplusplus.com/reference/string/stod/>

(дата обращения: 14.10.2020)