**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: Наследование, полиморфизм

Студент: Ильминский Никита

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

В данной лабораторной работе мне необходимо разработать классы 8-угольника, треугольника и квадрата; классы должны наследоваться от базового класса Figure и поддерживать набор общих методов: вычисление геометрического центра фигуры, вывод в стандартный поток вывода std::cout координат вершин фигуры и вычисление площади фигуры.

Написанная программа должна позволять: вводить из стандартного ввода std::cin фигуры 8-угольника, треугольника или квадрата, сохранять созданные фигуры в динамический массив std::vector<Figure\*>, вызывать для всего массива общие функции (распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь), вычислять общую площадь фигур в массиве, удалять из массива фигуру по индексу.

1. Описание программы

Для выполнения данной задачи в файле main.cpp создан базовый класс Figure и его производные классы Octagon, Triangle и Square; все три класса наследуют public методы базового класса: виртуальные функции для получения геом. центра, координат и площади каждой фигуры и функция для получения имени класса. В производных классах создаются конструкторы для корректного создания объектов класса и переписываются (override) виртуальные функции для правильного вывода для каждой фигуры (для каждой фигуры значения площади, геом. центр и координаты вычисляются по-разному); в классе Octagon для построения конструктор принимает координаты центра фигуры и радиус описанной окружности (больший 0), конструктор класса Triangle принимает три точки и проверяет их правильность (точки не должны лежать на 1 прямой, то есть площадь не может быть равна 0), конструктор класса Square принимает две противоположные точки диагонали (обязательно две разные) и на основе геом. свойств квадрата вычисляет две остальные точки. Для удобного ввода координат фигур создана структура point, хранящая значение абсциссы и ординаты точки.

Для вызова функций для различных фигур создается вектор типа Figure\*: в векторе хранятся указатели базового класса Figure, хранящие ссылки на производные классы фигур; при помощи обхода этого вектора и операции получения доступа к методам классов производится вызов необходимых функций. В функции int main() реализован интерфейс работы с программой: при помощи функции menu() выводится список доступных операций; пользователь вводит операцию и программа вызывает необходимые функции (вставка фигуры, удаление по индексу, вывод площадей, координат, геом. центра, выход из программы).

1. Набор testcases
2. T 3.5 34 0 0 12 1.5
3. S 0 0 1 1
4. S 0 0 0.5 1
5. O 1.5 0 1.5
6. T 0 0 3 3 40.32 40.32
7. O 3.4456 4 -1
8. S 1.34 1 1.34 1
9. Результаты выполнения тестов

root@DESKTOP-5B8SU4Q:~/labs/3sem/oop/3# ./oop\_exercise\_03

List of operations with figures:

0. "menu" to print this menu;

1. "add" to add a new figure;

2. "center" to get the geometric center for each figure;

3. "vertices" to get the vertices of each figure;

4. "area" to get the area of each figure;

5. "total" to get the total area of all figures;

6. "delete" to erase a figure by an index;

7. "quit" to exit the program;

Enter the needed operation:

add

Enter O for an octagon, T for a triangle, S for a square:

T

Enter three vertices for a triangle: 3.5 34 0 0 12 1.5

Triangle added at index 0

add

Enter O for an octagon, T for a triangle, S for a square:

S

Enter two opposite vertices of square: 0 0 1 1

Square added at index 1

add

Enter O for an octagon, T for a triangle, S for a square:

S

Enter two opposite vertices of square: 0 0 0.5 1

Square added at index 2

add

Enter O for an octagon, T for a triangle, S for a square:

O

Enter the coordinate of a center and the radius of a circle: 1.5 0 1.5

Octagon added at index 3

center

Center of Triangle is (5.16667, 11.8333)

Center of Square is (0.5, 0.5)

Center of Square is (0.25, 0.5)

Center of Octagon is (1.5, 0)

vertices

Triangle vertices are: (3.5, 34), (0, 0), (12, 1.5)

Square vertices are: (0, 0), (0, 1), (1, 1), (1, 0)

Square vertices are: (0, 0), (-0.25, 0.75), (0.5, 1), (0.75, 0.25)

Octagon vertices are: (1.5, 1.5), (2.56066, 1.06066), (3, 0), (2.56066, -1.06066), (1.5, -1.5), (0.43934, -1.06066), (0, 0), (0.43934, 1.06066)

area

The area of Triangle is 201.375

The area of Square is 1

The area of Square is 0.625

The area of Octagon is 6.36396

total

The total area of all figures is: 209.364

delete

Enter the index of a figure you want to erase: 1

Element erased.

total

The total area of all figures is: 208.364

area

The area of Triangle is 201.375

The area of Square is 0.625

The area of Octagon is 6.36396

quit

root@DESKTOP-5B8SU4Q:~/labs/3sem/oop/3# ./oop\_exercise\_03

List of operations with figures:

0. "menu" to print this menu;

1. "add" to add a new figure;

2. "center" to get the geometric center for each figure;

3. "vertices" to get the vertices of each figure;

4. "area" to get the area of each figure;

5. "total" to get the total area of all figures;

6. "delete" to erase a figure by an index;

7. "quit" to exit the program;

Enter the needed operation:

add

Enter O for an octagon, T for a triangle, S for a square:

T

Enter three vertices for a triangle: 0 0 3 3 40.32 40.32

Incorrect vertices for a triangle! The area is 0

root@DESKTOP-5B8SU4Q:~/labs/3sem/oop/3# ./oop\_exercise\_03

List of operations with figures:

0. "menu" to print this menu;

1. "add" to add a new figure;

2. "center" to get the geometric center for each figure;

3. "vertices" to get the vertices of each figure;

4. "area" to get the area of each figure;

5. "total" to get the total area of all figures;

6. "delete" to erase a figure by an index;

7. "quit" to exit the program;

Enter the needed operation:

add

Enter O for an octagon, T for a triangle, S for a square:

O

Enter the coordinate of a center and the radius of a circle: 3.4456 4 -1

Incorrect radius for an octagon! Radius is <= 0

root@DESKTOP-5B8SU4Q:~/labs/3sem/oop/3# ./oop\_exercise\_03

List of operations with figures:

0. "menu" to print this menu;

1. "add" to add a new figure;

2. "center" to get the geometric center for each figure;

3. "vertices" to get the vertices of each figure;

4. "area" to get the area of each figure;

5. "total" to get the total area of all figures;

6. "delete" to erase a figure by an index;

7. "quit" to exit the program;

Enter the needed operation:

add

Enter O for an octagon, T for a triangle, S for a square:

S

Enter two opposite vertices of square: 1.34 1 1.34 1

Incorrect vertices for a square! Coordinates are equal

1. Листинг программы

//Ильминский Никита, М8О-207Б-18

//Вариант 8: 8-угольник, Треугольник, Квадрат

/\*Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure.

Фигуры являются фигурами вращения.

Все классы должны поддерживать набор общих методов:

Вычисление геометрического центра фигуры;

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

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;

\*/

#include <iostream>

#include <cmath>

#include <vector>

#include <string>

struct point {

float x;

float y;

};

std::ostream& operator<< (std::ostream& out, const point& p) {

out << "(" << p.x << ", " << p.y << ")";

return out;

}

class Figure {

protected:

std::string name;

public:

std::string getName() {

return name;

}

virtual point GeometricCenter() = 0;

virtual void Vertices() = 0;

virtual float Area() = 0;

};

class Octagon: public Figure {

point p1, p2, p3, p4, p5, p6, p7, p8;

float a;

public:

Octagon() {

name = "Octagon";

}

Octagon(const point p, const float r) {

if(r <= 0) {

std::cout << "Incorrect radius for an octagon! Radius is <= 0\n";

exit(1);

}

name = "Octagon";

float r2 = r / sqrt(2);

p1.x = p.x; p1.y = p.y + r;

p2.x = p.x + r2; p2.y = p.y + r2;

p3.x = p.x + r; p3.y = p.y;

p4.x = p.x + r2; p4.y = p.y - r2;

p5.x = p.x; p5.y = p.y - r;

p6.x = p.x - r2; p6.y = p.y - r2;

p7.x = p.x - r; p7.y = p.y;

p8.x = p.x - r2; p8.y = p.y + r2;

a = sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));

}

~Octagon() = default;

point GeometricCenter() override {

point center;

center.x = (p1.x + p5.x) / 2;

center.y = (p1.y + p5.y) / 2;

return center;

}

void Vertices() override {

std::cout << "Octagon vertices are: " << p1 << ", " << p2 << ", "

<< p3 << ", " << p4 << ", " << p5 << ", " << p6 << ", " << p7 << ", " << p8 << std::endl;

}

float Area() override {

return 2 \* (1 + sqrt(2)) \* pow(a, 2);

}

};

class Triangle: public Figure {

point p1, p2, p3;

float a, b, c;

public:

Triangle() {

name = "Triangle";

}

Triangle(const point v1, const point v2, const point v3) {

name = "Triangle";

p1 = v1;

p2 = v2;

p3 = v3;

a = sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));

b = sqrt(pow(p2.x - p3.x, 2) + pow(p2.y - p3.y, 2));

c = sqrt(pow(p3.x - p1.x, 2) + pow(p3.y - p1.y, 2));

if(Area() == 0) {

std::cout << "Incorrect vertices for a triangle! The area is 0\n";

exit(1);

}

}

~Triangle() = default;

point GeometricCenter() override {

point center;

center.x = (p1.x + p2.x + p3.x) / 3;

center.y = (p1.y + p2.y + p3.y) / 3;

return center;

}

void Vertices() override {

std::cout << "Triangle vertices are: " << p1 << ", " << p2 << ", " << p3 << std::endl;

}

float Area() override {

float p = (a + b + c) / 2;

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

};

class Square: public Figure {

point p1, p2, p3, p4;

float a;

public:

Square() {

name = "Square";

}

Square(const point v1, const point v2) {

if(v1.x == v2.x && v1.y == v2.y) {

std::cout << "Incorrect vertices for a square! Coordinates are equal\n";

exit(1);

}

name = "Square";

p1 = v1;

p3 = v2;

a = sqrt((pow(p3.x - p1.x, 2) + pow(p3.y - p1.y, 2)) / 2);

point c = GeometricCenter();

point perp;

perp.x = (p3.y - p1.y) / 2;

perp.y = (p1.x - p3.x) / 2;

p2.x = c.x - perp.x;

p2.y = c.y - perp.y;

p4.x = c.x + perp.x;

p4.y = c.y + perp.y;

}

point GeometricCenter() override {

point center;

center.x = (p1.x + p3.x) / 2;

center.y = (p1.y + p3.y) / 2;

return center;

}

void Vertices() override {

std::cout << "Square vertices are: " << p1 << ", " << p2 << ", " << p3 << ", " << p4 << std::endl;

}

float Area() override {

return a \* a;

}

};

inline void menu() {

std::cout << "List of operations with figures:\n";

std::cout << "0. \"menu\" to print this menu;\n";

std::cout << "1. \"add\" to add a new figure;\n";

std::cout << "2. \"center\" to get the geometric center for each figure;\n";

std::cout << "3. \"vertices\" to get the vertices of each figure;\n";

std::cout << "4. \"area\" to get the area of each figure;\n";

std::cout << "5. \"total\" to get the total area of all figures;\n";

std::cout << "6. \"delete\" to erase a figure by an index;\n";

std::cout << "7. \"quit\" to exit the program;\n";

std::cout << "Enter the needed operation:\n";

}

int main()

{

std::vector<Figure\*> v;

Figure\* f = nullptr;

std::string action;

menu();

char figure;

while(std::cin >> action) {

if(action == "menu") {

menu();

}

if(action == "add") {

std::cout << "Enter O for an octagon, T for a triangle, S for a square:\n";

std::cin >> figure;

point p1, p2, p3;

float r;

switch(figure) {

case 'O':

std::cout << "Enter the coordinate of a center and the radius of a circle: ";

std::cin >> p1.x >> p1.y >> r;

f = new Octagon(p1, r);

v.push\_back(f);

std::cout << "Octagon added at index " << v.size() - 1 << std::endl;

break;

case 'T':

std::cout << "Enter three vertices for a triangle: ";

std::cin >> p1.x >> p1.y >> p2.x >> p2.y >> p3.x >> p3.y;

f = new Triangle(p1, p2, p3);

v.push\_back(f);

std::cout << "Triangle added at index " << v.size() - 1 << std::endl;

break;

case 'S':

std::cout << "Enter two opposite vertices of square: ";

std::cin >> p1.x >> p1.y >> p2.x >> p2.y;

f = new Square(p1, p2);

v.push\_back(f);

std::cout << "Square added at index " << v.size() - 1 << std::endl;

break;

default:

std::cout << "\'" << figure << "\' is invalid figure\n";

break;

}

}

if(action == "center") {

for(size\_t i = 0; i < v.size(); i++) {

std::cout << "Center of " << v[i]->getName() << " is " << v[i]->GeometricCenter() << std::endl;

}

}

if(action == "vertices" || action == "vert") {

for(size\_t i = 0; i < v.size(); i++) {

v[i]->Vertices();

}

}

if(action == "area") {

for(size\_t i = 0; i < v.size(); i++) {

std::cout << "The area of " << v[i]->getName() << " is " << v[i]->Area() << std::endl;

}

}

if(action == "total") {

float s = 0;

for(size\_t i = 0; i < v.size(); i++) {

s += v[i]->Area();

}

std::cout << "The total area of all figures is: " << s << std::endl;

}

if(action == "delete") {

size\_t idx;

std::cout << "Enter the index of a figure you want to erase: ";

std::cin >> idx;

if(idx >= v.size()) {

std::cout << "Wrong index! Vector size is " << v.size() << std::endl;

continue;

}

v.erase(v.begin() + idx);

std::cout << "Element erased.\n";

}

if(action == "quit") {

delete f;

break;

}

}

return 0;

}

1. Вывод

Наследование – полезный механизм для реализации множества классов, похожих по своим функциям и свойствам на базовый. Наследование позволяет не переписывать код для каждого нового класса, используя реализацию базового класса.

При наследовании при помощи виртуальных функций реализуется полиморфизм — способность принимать разные формы. Также полиморфизм в C++ может быть реализован при помощи перегрузки функций и перегрузки операторов. Виртуальные функции позволяют переопределять поведение методов, что определяет один из принципов полиморфизма “один интерфейс – разные реализации”.

Благодаря этому полиморфизм помогает работать с разными объектами, имеющими сходные свойства - например, как в нашем случае, мы можем создавать огромную коллекцию различных фигур и для каждой из них вызывать однотипные функции, реализация которых будет отличаться.

СПИСОК ЛИТЕРАТУРЫ

1. C++ Programming Tutorials Playlist - thenewboston [Электронный ресурс].  
   URL: <https://www.youtube.com/playlist?list=PLAE85DE8440AA6B83>  
   (дата обращения: 12.10.2019).
2. Overloading input/output operators in C++ [Электронный ресурс].

URL: https://medium.com/@nonuruzun/overloading-input-output-operators-in-c-a2a74c5dda8a (дата обращения: 13.10.2019).

1. Система вопросов и ответов Stack Overflow [Электронный ресурс]. URL: <https://stackoverflow.com> (дата обращения 13.10.2019).