**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: Киреев Александр Константинович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Вариант 8.

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 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;

1. **Описание программы**

В программе реализован родительский класс Figure от которого наследуются классы Triangle, Square и Octagon. Родительский класс имеет 3 чисто виртуальных метода: getCenter(), print(), getArea(). В родительский класс встроен класс Dot, для удобства работы с точками. Также класс Figure имеет виртуальный деструктор, поле \_type и метод для получения типа фигуры для удобства работы с фигурами. Дочерние классы имеют перегруженные методы родительского класса, а также конструктор, который принимает поток ввода и конструирует объект из полученных из потока данных. Метод getArea() получает площадь фигуры по радиусу описанной окружности. Метод print() нуждается в координатах вершин, которые вычисляются в методе getCoords(). Зная координаты центра фигуры, координаты ее вершины, мы можем найти полярный угол и угол поворота, на который нужно отступить от заданной вершины, чтобы получить следующую вершину. В файле main.cpp мы имеем парсер команд для работы пользователя с программой, а также функции applyToVec(const std::vector<Figure\*>& vec), которая вычисляет всю информацию о каждой фигуре в векторе, и totalArea(const std::vector<Figure\*>& vec), которая считает общую площадь фигур в векторе.  Проект был сделан в ОС macOS, в редакторе VSCode.

1. **Набор тестов**

Тест 1:

wrong cmd

2 2

0 0 1 1

2 3

0 0 0 1

1 0

6

3

4

5 1

6

8

Тест 1 проверяет реакцию программу на не валидную команду, также проверяет работоспособность основных функций. В нем мы вводим квадрат и треугольник, выводим квадрат на экран, выводим весь вектор (6), получаем всю информацию о фигурах (3), считаем общую площадь (4), удаляем треугольник из вектора (5 1), проверяем, что все удалилось и заканчиваем программу.

Тест 2:

2 2

0 0 -1 -1

2 1

0 0 0 1

2 10

2 2

2 2 4 4

6

4

1 2

5 2

1 2

6

3

4

8

В тесте 2 мы вводим два квадрата и восьмиугольник. Также в этом тесте мы проверяем программу на устойчивость - подаем ей на ввод не валидный индекс. На этих фигурах мы также, как и в тесте 1 тестируем функции подсчета общей площади, вывод всего вектора площадей, удаление из вектора.

1. **Результаты выполнения тестов**

Тест 1:

Command -- Description

1 [idx] -- Display figure by index in vector

2 [1..3] -- Add figure (type 1 - oct, 2 - sq, 3 - tri)

3 -- Apply base methods to the entire vector

4 -- Calculate total area of figures in vector

5 [idx] -- Delete figure by index from vector

6 -- Display vector

7 -- Display help

8 -- End program

Invalid command.

Invalid command.

[ (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000), (1.000, -1.000) ]

=========START OF VECTOR=========

[ (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000), (1.000, -1.000) ]

[ (0.000, 1.000), (-0.866, -0.500), (0.866, -0.500) ]

==========END OF VECTOR==========

>>Next figure is Square:

Center: (0.000, 0.000)

Figure coords: [ (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000), (1.000, -1.000) ]

Area: 4.000

>>Next figure is Triangle:

Center: (0.000, 0.000)

Figure coords: [ (0.000, 1.000), (-0.866, -0.500), (0.866, -0.500) ]

Area: 1.299

5.299

=========START OF VECTOR=========

[ (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000), (1.000, -1.000) ]

==========END OF VECTOR==========

Тест 2:

Command -- Description

1 [idx] -- Display figure by index in vector

2 [1..3] -- Add figure (type 1 - oct, 2 - sq, 3 - tri)

3 -- Apply base methods to the entire vector

4 -- Calculate total area of figures in vector

5 [idx] -- Delete figure by index from vector

6 -- Display vector

7 -- Display help

8 -- End program

Invalid type.

=========START OF VECTOR=========

[ (1.000, -1.000), (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000) ]

[ (0.000, 1.000), (-0.707, 0.707), (-1.000, 0.000), (-0.707, -0.707), (-0.000, -1.000), (0.707, -0.707), (1.000, -0.000), (0.707, 0.707) ]

[ (4.000, 4.000), (0.000, 4.000), (-0.000, 0.000), (4.000, -0.000) ]

==========END OF VECTOR==========

22.828

[ (4.000, 4.000), (0.000, 4.000), (-0.000, 0.000), (4.000, -0.000) ]

Invalid index.

=========START OF VECTOR=========

[ (1.000, -1.000), (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000) ]

[ (0.000, 1.000), (-0.707, 0.707), (-1.000, 0.000), (-0.707, -0.707), (-0.000, -1.000), (0.707, -0.707), (1.000, -0.000), (0.707, 0.707) ]

==========END OF VECTOR==========

>>Next figure is Square:

Center: (0.000, 0.000)

Figure coords: [ (1.000, -1.000), (1.000, 1.000), (-1.000, 1.000), (-1.000, -1.000) ]

Area: 4.000

>>Next figure is Octagon:

Center: (0.000, 0.000)

Figure coords: [ (0.000, 1.000), (-0.707, 0.707), (-1.000, 0.000), (-0.707, -0.707), (-0.000, -1.000), (0.707, -0.707), (1.000, -0.000), (0.707, 0.707) ]

Area: 2.828

6.828

1. **Листинг программы**

**main.cpp**

// Киреев Александр Константинович 206

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include "octagon.hpp"

#include "square.hpp"

#include "triangle.hpp"

// функция, которая вызывает для каждого элемента массива его 3 базовых метода

void applyToVec(const std::vector<Figure\*>& vec) {

for (auto v : vec) {

std::cout << ">>Next figure is " << v->getType() << ": " << std::endl;

std::cout << "Center: " << v->getCenter() << std::endl;

std::cout << "Figure coords: ";

v->print(std::cout);

std::cout << std::endl;

std::cout << "Area: " << v->getArea() << std::endl;

}

}

// считает суммарную площадь всех фигур в векторе

double totalArea(const std::vector<Figure\*>& vec) {

double result = 0.0;

for (auto v : vec) {

result += v->getArea();

}

return result;

}

// мануал

void help() {

std::cout << "Command -- Description\n" <<

"1 [idx] -- Display figure by index in vector\n" <<

"2 [1..3] -- Add figure (type 1 - oct, 2 - sq, 3 - tri)\n" <<

"3 -- Apply base methods to the entire vector\n" <<

"4 -- Calculate total area of figures in vector\n" <<

"5 [idx] -- Delete figure by index from vector\n" <<

"6 -- Display vector\n" <<

"7 -- Display help\n" <<

"8 -- End program" << std::endl;

}

void mainLoop() {

int command = 0, idx = 0, type = 0;

std::vector<Figure\*> vec;

Figure\* f;

std::string s;

while (std::cout << "Cmd: " && std::cin >> s) {

if (s.length() > 1) {

std::cout << "Invalid command." << std::endl;

continue;

}

std::stringstream ss(s);

ss >> command;

if (command == 8)

break;

switch (command) {

case 1:

std::cin >> idx;

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

std::cout << "Invalid index." << std::endl;

break;

}

vec[idx]->print(std::cout);

std::cout << std::endl;

break;

case 2:

std::cin >> type;

switch (type) {

case 1:

f = new Octagon(std::cin);

vec.push\_back(f);

break;

case 2:

f = new Square(std::cin);

vec.push\_back(f);

break;

case 3:

f = new Triangle(std::cin);

vec.push\_back(f);

break;

default:

std::cout << "Invalid type." << std::endl;

break;

}

break;

case 3:

applyToVec(vec);

break;

case 4:

std::cout << totalArea(vec) << std::endl;

break;

case 5:

std::cin >> idx;

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

std::cout << "Invalid index." << std::endl;

break;

}

f = vec[idx];

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

// освобождаем память, чтоб не утеклало по памяти

delete f;

break;

case 6:

std::cout << "=========START OF VECTOR=========" << std::endl;

for (auto v : vec) {

v->print(std::cout);

std::cout << std::endl;

}

std::cout << "==========END OF VECTOR==========" << std::endl;

break;

case 7:

help();

break;

default:

std::cout << "Invalid command!" << std::endl;

break;

}

}

// освобождаем память в векторе, чтобы не утекало

for (auto v : vec) {

delete v;

}

}

int main() {

help();

mainLoop();

return 0;

}

**dot.hpp**

// Киреев Александр Константинович 206

#pragma once

#include <iostream>

struct Dot {

public:

double X, Y;

Dot();

Dot(double x, double y);

Dot(const Dot& d);

~Dot() = default;

};

std::istream& operator>>(std::istream& is, Dot& d);

std::ostream& operator<<(std::ostream& os, const Dot& d);

**dot.cpp**

// Киреев Александр Константинович 206

#include "dot.hpp"

Dot::Dot() {

X = 0.0;

Y = 0.0;

}

Dot::Dot(double x, double y) {

X = x;

Y = y;

}

Dot::Dot(const Dot& d) {

this->X = d.X;

this->Y = d.Y;

}

std::istream& operator>>(std::istream& is, Dot& d) {

is >> d.X >> d.Y;

return is;

}

std::ostream& operator<<(std::ostream& os, const Dot& d) {

std::cout.precision(3);

os << "(" << std::fixed << d.X << ", " << std::fixed << d.Y << ")";

return os;

}

**figure.hpp**

// Киреев Александр Константинович 206

#pragma once

#include "dot.hpp"

#include <string>

class Figure {

public:

virtual Dot getCenter() const = 0;

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

virtual double getArea() const = 0;

// добавим метод getType(), чтобы при выводе видить, что за фигура перед нами

virtual std::string getType() const { return \_type; }

// виртуальный деструктор, чтобы корректно вызывались деструкторы дочерних классов

virtual ~Figure() = default;

protected:

std::string \_type;

};

**octagon.cpp**

// Киреев Александр Константинович 206

#include "octagon.hpp"

#include "cmath"

#define \_USE\_MATH\_DEFINES

Octagon::Octagon(const Dot& center, const Dot& d): \_center(center), \_d(d) {

\_type = "Octagon";

}

Octagon::Octagon(const Octagon& o) {

this->\_center = o.\_center;

this->\_d = o.\_d;

this->\_type = o.\_type;

}

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

is >> this->\_center >> this->\_d;

\_type = "Octagon";

}

Dot Octagon::getCenter() const {

return this->\_center;

}

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

std::vector<Dot> dots;

this->getCoords(dots);

os << "[ " << dots[0];

for (int i = 1; i < dots.size(); ++i) {

os << ", " << dots[i];

}

os << " ]";

}

// будем считать площадь через описанную окружность, тк порядок точек фигуры не влияет на ее результат

double Octagon::getArea() const {

double r = sqrt((this->\_d.X - this->\_center.X) \* (this->\_d.X - this->\_center.X) +

(this->\_d.Y - this->\_center.Y) \* (this->\_d.Y - this->\_center.Y));

return 2.0 \* sqrt(2.0) \* r \* r;

}

std::string Octagon::getType() const {

return \_type;

}

void Octagon::getCoords(std::vector<Dot>& dots) const {

double r = sqrt((this->\_d.X - this->\_center.X) \* (this->\_d.X - this->\_center.X) +

(this->\_d.Y - this->\_center.Y) \* (this->\_d.Y - this->\_center.Y));

double lenX = this->\_d.X - this->\_center.X;

double lenY = this->\_d.Y - this->\_center.Y;

double phi = acos(lenX / r);

if (lenX < 0) {

if (lenY < 0)

phi = -M\_PI + phi;

} else {

if (lenY < 0)

phi = -phi;

}

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

double angle = 2.0 \* M\_PI \* i / 8.0;

dots.push\_back(Dot(r \* cos(angle + phi) + this->\_center.X, r \* sin(angle + phi) + this->\_center.Y));

}

}

std::ostream& operator<<(std::ostream& os, const Octagon& o) {

o.print(os);

return os;

}

**octagon.hpp**

// Киреев Александр Константинович 206

#pragma once

#include "figure.hpp"

#include "dot.hpp"

#include <iostream>

#include <string>

#include <vector>

class Octagon: public Figure {

public:

Octagon() = default;

Octagon(const Dot& center, const Dot& d);

Octagon(const Octagon& o);

Octagon(std::istream& is);

Dot getCenter() const override;

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

double getArea() const override;

std::string getType() const override;

void getCoords(std::vector<Dot>& dots) const;

~Octagon() = default;

private:

Dot \_center, \_d;

};

std::ostream& operator<<(std::ostream& os, const Octagon& o);

**triangle.cpp**

// Киреев Александр Константинович 206

#include "triangle.hpp"

#include "cmath"

#define \_USE\_MATH\_DEFINES

Triangle::Triangle(const Dot& center, const Dot& d) : \_center(center), \_d(d) {

\_type = "Triangle";

}

Triangle::Triangle(const Triangle& t) {

this->\_center = t.\_center;

this->\_d = t.\_d;

this->\_type = t.\_type;

}

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

is >> this->\_center >> this->\_d;

\_type = "Triangle";

}

Dot Triangle::getCenter() const {

return this->\_center;

}

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

std::vector<Dot> dots;

this->getCoords(dots);

os << "[ " << dots[0];

for (int i = 1; i < dots.size(); ++i) {

os << ", " << dots[i];

}

os << " ]";

}

// будем считать площадь через описанную окружность, тк порядок точек фигуры не влияет на ее результат

double Triangle::getArea() const {

double r = sqrt((this->\_d.X - this->\_center.X) \* (this->\_d.X - this->\_center.X) +

(this->\_d.Y - this->\_center.Y) \* (this->\_d.Y - this->\_center.Y));

return 3.0 \* sqrt(3.0) \* r \* r / 4.0;

}

std::string Triangle::getType() const {

return \_type;

}

void Triangle::getCoords(std::vector<Dot>& dots) const {

double r = sqrt((this->\_d.X - this->\_center.X) \* (this->\_d.X - this->\_center.X) +

(this->\_d.Y - this->\_center.Y) \* (this->\_d.Y - this->\_center.Y));

double lenX = this->\_d.X - this->\_center.X;

double lenY = this->\_d.Y - this->\_center.Y;

double phi = acos(lenX / r);

if (lenX < 0) {

if (lenY < 0)

phi = -M\_PI + phi;

} else {

if (lenY < 0)

phi = -phi;

}

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

double angle = 2.0 \* M\_PI \* i / 3.0;

dots.push\_back(Dot(r \* cos(angle + phi) + this->\_center.X, r \* sin(angle + phi) + this->\_center.Y));

}

}

std::ostream& operator<<(std::ostream& os, const Triangle& t) {

t.print(os);

return os;

}

**triangle.hpp**

// Киреев Александр Константинович 206

#pragma once

#include "figure.hpp"

#include "dot.hpp"

#include <iostream>

#include <string>

#include <vector>

class Triangle: public Figure {

public:

Triangle() = default;

Triangle(const Dot& center, const Dot& d);

Triangle(const Triangle& t);

Triangle(std::istream& is);

Dot getCenter() const override;

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

double getArea() const override;

std::string getType() const override;

void getCoords(std::vector<Dot>& dots) const;

~Triangle() = default;

private:

Dot \_center, \_d;

};

std::ostream& operator<<(std::ostream& os, const Triangle& t);

**square.cpp**

// Киреев Александр Константинович 206

#include "square.hpp"

#include "cmath"

#define \_USE\_MATH\_DEFINES

Square::Square(const Dot& center, const Dot& d) :

\_center(center), \_d(d) {

\_type = "Square";

}

Square::Square(const Square& s) {

this->\_center = s.\_center;

this->\_d = s.\_d;

this->\_type = s.\_type;

}

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

is >> this->\_center >> this->\_d;

\_type = "Square";

}

Dot Square::getCenter() const {

return this->\_center;

}

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

std::vector<Dot> dots;

this->getCoords(dots);

os << "[ " << dots[0];

for (int i = 1; i < dots.size(); ++i) {

os << ", " << dots[i];

}

os << " ]";

}

// будем считать площадь через описанную окружность, тк порядок точек фигуры не влияет на ее результат

double Square::getArea() const {

double r = sqrt((this->\_d.X - this->\_center.X) \* (this->\_d.X - this->\_center.X) +

(this->\_d.Y - this->\_center.Y) \* (this->\_d.Y - this->\_center.Y));

return 2.0 \* r \* r;

}

std::string Square::getType() const {

return \_type;

}

void Square::getCoords(std::vector<Dot>& dots) const {

double r = sqrt((this->\_d.X - this->\_center.X) \* (this->\_d.X - this->\_center.X) +

(this->\_d.Y - this->\_center.Y) \* (this->\_d.Y - this->\_center.Y));

double lenX = this->\_d.X - this->\_center.X;

double lenY = this->\_d.Y - this->\_center.Y;

double phi = acos(lenX / r);

if (lenX < 0) {

if (lenY < 0)

phi = -M\_PI + phi;

} else {

if (lenY < 0)

phi = -phi;

}

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

double angle = 2.0 \* M\_PI \* i / 4.0;

dots.push\_back(Dot(r \* cos(angle + phi) + this->\_center.X, r \* sin(angle + phi) + this->\_center.Y));

}

}

std::ostream& operator<<(std::ostream& os, const Square& s) {

s.print(os);

return os;

}

**square.hpp**

// Киреев Александр Константинович 206

#pragma once

#include "figure.hpp"

#include "dot.hpp"

#include <iostream>

#include <string>

#include <vector>

class Square: public Figure {

public:

Square() = default;

Square(const Dot& center, const Dot& d);

Square(const Square& s);

Square(std::istream& is);

Dot getCenter() const override;

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

double getArea() const override;

std::string getType() const override;

void getCoords(std::vector<Dot>& dots) const;

~Square() = default;

private:

Dot \_center, \_d;

};

std::ostream& operator<<(std::ostream& os, const Square& s);

1. **Вывод**

Наследование в языке C++ - очень мощный инструмент, одна из основ ООП. С его помощью можно организовывать семейства классов, что позволит в некоторых местах избежать излишнего копирования кода, также это логически делает код более понятным, защищенным от ошибок программиста. Но нужно знать меру наследованию, использовать его с умом, избегая всяческих множественных наследований, которые могут привести к наличию общего родителя.

1. **Список литературы**

“Язык программирования C++. Краткий курс”, Бьярне Страуструп, 2-е издание, (перевод Игоря Красикова), бумага офсетная-белая, твердый переплет, 320 стр., ISBN 978-5-907144-12-5.