Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 3: «Наследование, полиморфизм»

Группа:	М8О-206Б-18, №14
Студент:	Орозбакиев Эмиль Данияpович
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	

Москва, 2019

1. Задание

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 14: 5-уголник, 6-угольник, 8-угольник

2. Адрес репозитория на GitHub

https://github.com/p0kemo4ik/oop_exercise_03

3. Код программы на C++

main.cpp

```
#include<iostream>
#include"figs.hpp"
#include<locale>
int getOption() {
   int Menu;
   std::cout << "1. Ввести фигуру" << std::endl;
   std::cout << "2. Вычислить центр фигуры по индексу" << std::endl;
   std::cout << "3. Вычислить площадь фигуры по индексу" << std::endl;
   std::cout << "4. Распечатать координаты фигуры по индексу" <<
std::endl;
   std::cout << "5. Вычислить общую площадь всех фигур" << std::endl;
   std::cout << "6. Удалить фигуру по индексу" << std::endl;
```

```
std::cout << "7. Задать координаты фигуры по количеству вершин, радиусу, центру фигуры и углу вращения" << std::endl;

    std::cin >> Menu;
    return Menu;
}
int whatFigure() {
    int Menu;
    std::cout << "1. Ввести 5-угольник" << std::endl;
    std::cout << "2. Ввести 6-угольник" << std::endl;
    std::cout << "3. Ввести 8-угольник" << std::endl;
    std::cin >> Menu;
    return Menu;
}
int main() {
    setlocale(LC_ALL, "rus");
    int Menu_1,Menu_2, Index;
    double SummaryArea = 0;
    Figure* f;
    std::vector<Figure*> Figures;
    while (true) {
        switch (Menu_1 = getOption()) {
            case 1:
                switch (Menu_2 = whatFigure()) {
                    case 1:
                        f = new Pentagon{ std::cin };
                        break;
                    case 2:
                        f = new Hexagon{ std::cin };
                        break;
                    case 3:
                        f = new Octagon(std::cin);
                        break;
                }
                Figures.push_back(f);
                break;
            case 2:
                std::cout << "Введите индекс: ";
                std::cin >> Index;
                if (Figures[Index] != nullptr)
                    std::cout << "Центр фигуры по индексу " << Index << ": " << (*Figures[Index]).center() << std::endl;
                break;
            case 3:
                std::cout << "Введите индекс: ";
```

```
            std::cin >> Index;
            if (Figures[Index] != nullptr)
                std::cout << "Площадь фигуры по индексу " << Index << ": " <<
(*Figures[Index]).square() << std::endl;
            break;
        case 4:
            std::cout << "Введите индекс: ";
            std::cin >> Index;
            std::cout << "Координты фигуры по индексу " << Index << ": ";
            (*Figures[Index]).printCords();
            std::cout << std::endl;
            continue;
        case 5:
            for (int i = 0; i < (int)Figures.size(); i++)
                if (Figures[i] != nullptr) {
                    (*Figures[i]).printCords();
                    std::cout << std::endl;
                    std::cout << "Area: " << (*Figures[i]).square() << std::endl;
                    std::cout << "Center: " << (*Figures[i]).center() << std::endl;
                }
            std::cout << "Общая площадь фигур: " << SummaryArea <<
std::endl;
            break;
        case 6:
            std::cout << "Введите индекс: ";
            std::cin >> Index;
            std::swap(Figures[Figures.size() - 1], Figures[Index]);
            delete Figures[Figures.size() - 1];
            Figures.pop_back();
            break;
case 7: {
    std::cout << "Введите R, n, x0, y0, phi" << std::endl;
    double x, y, R, phi;
    int n;
    std::cin >> R >> n >> x >> y >> phi;
    std::vector<Vertex> z0;
    for (int i = 0; i < n; i++) {
        z0.push_back(Vertex(x + R * cos(phi + 2 * M_PI * i / n),
                            y + R * (float) sin(phi + 2 * M_PI * i / n)));
    }
    for (int i = 0; i < n; i++) {
        std::cout << z0[i].x <<" " << z0[i].y<<std::endl;
    }
    std::cout << std::endl;
}
```

```
            default:
                for (int i = 0; i < (int)Figures.size(); i++) {
                    delete Figures[i];
                    Figures[i] = nullptr;
                }
                return 0;
        }
    }
    return 0;
}
```

vertex.hpp

```
#include<iostream>
class Vertex {
public:
    double x, y;
    Vertex();
    Vertex(double _x, double _y);
    Vertex& operator+=(const Vertex& b);
    Vertex& operator-=(const Vertex& b);
    friend std::ostream& operator<< (std::ostream &out, const Vertex &point);
};
Vertex operator+ (const Vertex &a, const Vertex& b);
Vertex operator- (const Vertex &a, const Vertex& b);
Vertex operator/ (const Vertex &a, const double& b);
double distance(const Vertex &a, const Vertex& b);
double vector_product(const Vertex& a, const Vertex& b);
```

vertex.cpp

```
#pragma once
#include"vertex.hpp"
#include<cmath>
Vertex::Vertex(): x(0),y(0) {}
Vertex::Vertex(double _x, double _y): x(_x), y(_y) {}
Vertex& Vertex::operator+=(const Vertex& b) {
    x += b.x;
    y += b.y;
    return *this;
}
Vertex& Vertex::operator-=(const Vertex& b) {
    x -= b.x;
    y -= b.y;
    return *this;
```

```
}
Vertex operator+(const Vertex &a, const Vertex& b) {
    return Vertex(a.x + b.x, a.y + b.y);
}
Vertex operator-(const Vertex &a, const Vertex& b) {
    return Vertex(a.x - b.x, a.y - b.y);
}
Vertex operator/(const Vertex &a, const double& b) {
    return Vertex(a.x / b, a.y / b);
}
double distance(const Vertex &a, const Vertex& b) {
    return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));
}
double vector_product(const Vertex& a, const Vertex& b) {
    return a.x*b.y - b.x*a.y;
}
std::ostream& operator<< (std::ostream &out, const Vertex &point) {
    out << "[" << point.x << ", " << point.y << ']';
    return out;
}
```

figure.hpp

```
#include<iostream>
#include<vector>
#include"vertex.hpp"
class Figure {
public:
    virtual Vertex center() const = 0;
    virtual double square() const = 0;
    virtual void printCords() const = 0;
    //virtual ~Figure();
};
```

figs.hpp

```
#pragma once
#include<iostream>
#include"figure.hpp"
class Pentagon : public Figure {
private:
    Vertex Vertexs[5];
public:
    Pentagon();
    Pentagon(std::istream& in);
```

```
    Vertex center() const override;
    double square() const override;
    void printCords() const override;
};
class Hexagon : public Figure {
private:
    Vertex Vertexs[6];
public:
    Hexagon();
    Hexagon(std::istream &in);
    Vertex center() const override;
    double square() const override;
    void printCords() const override;
};
class Octagon : public Figure {
private:
    Vertex Vertexs[8];
public:
    Octagon();
    Octagon(std::istream &in);
    Vertex center() const override;
    double square() const override;
    void printCords() const override;
};
```

figs.cpp

```
#include<iostream>
#include"figs.hpp"
#include<cmath>
#include<cassert>
//Pentagon
Pentagon::Pentagon() {};
Pentagon::Pentagon(std::istream& in) {
    in >> Vertexs[0].x >> Vertexs[0].y >> Vertexs[1].x >> Vertexs[1].y >>
Vertexs[2].x >> Vertexs[2].y >> Vertexs[3].x >> Vertexs[3].y >> Vertexs[4].x >>
Vertexs[4].y;
}
Vertex Pentagon::center() const {
    Vertex res = Vertex();
    for (int i = 0; i < 5; i++)
        res += Vertexs[i];
    return res / 5;
}
double Pentagon::square() const  {
```

```
    double Area = 0;
    for (int i = 0; i < 5; i++) {
        Area += (Vertexs[i].x) * (Vertexs[(i + 1)%5].y) - (Vertexs[(i +
1)%5].x)*(Vertexs[i].y);
    }
    Area *= 0.5;
    return abs(Area);
}
void Pentagon::printCords() const {
    std::cout << "Pentagon: ";
    for (int i = 0; i < 5; i++)
        std::cout << Vertexs[i] << ' ';
    std::cout << '\b';
}
//Hexagon
Hexagon::Hexagon() {};
Hexagon::Hexagon(std::istream &in) {
    in >> Vertexs[0].x >> Vertexs[0].y >> Vertexs[1].x >> Vertexs[1].y >>
Vertexs[2].x >> Vertexs[2].y >> Vertexs[3].x
       >> Vertexs[3].y >> Vertexs[4].x >> Vertexs[4].y >> Vertexs[5].x >>
Vertexs[5].y;}
    Vertex Hexagon::center() const {
        Vertex res = Vertex();
        for (int i = 0; i < 6; i++)
            res += Vertexs[i];
        return res / 6;
    }
    double Hexagon::square() const  {
        double Area = 0;
        for (int i = 0; i < 6; i++) {
            Area += (Vertexs[i].x) * (Vertexs[(i + 1)%6].y) - (Vertexs[(i +
1)%6].x)*(Vertexs[i].y);
        }
        Area *= 0.5;
        return abs(Area);
    }
    void Hexagon::printCords() const {
        std::cout << "Hexagon: ";
        for (int i = 0; i < 6; i++)
            std::cout << Vertexs[i] << ' ';
        std::cout << '\b';
    }
//Hexagon
Octagon::Octagon(){};
Octagon::Octagon(std::istream &in) {
```

```
    in >> Vertexs[0].x >> Vertexs[0].y >> Vertexs[1].x >> Vertexs[1].y >>
Vertexs[2].x >> Vertexs[2].y >> Vertexs[3].x
        >> Vertexs[3].y >> Vertexs[4].x >> Vertexs[4].y >> Vertexs[5].x >>
Vertexs[5].y >> Vertexs[6].x >> Vertexs[6].y
                                                    >> Vertexs[7].x >> Vertexs[7].y;
}
    Vertex Octagon::center() const {
        Vertex res = Vertex();
        for (int i = 0; i < 8; i++)
            res += Vertexs[i];
        return res / 8;
    }
    double Octagon::square() const  {
        double Area = 0;
        for (int i = 0; i < 8; i++) {
            Area += (Vertexs[i].x) * (Vertexs[(i + 1)%8].y) - (Vertexs[(i +
1)%8].x)*(Vertexs[i].y);
        }
        Area *= 0.5;
        return abs(Area);
    }
    void Octagon::printCords() const {
        std::cout << "Octagon: ";
        for (int i = 0; i < 8; i++)
            std::cout << Vertexs[i] << ' ';
        std::cout << '\b';
    }
```

4. Результаты выполнения тестов

№	Фигура	Координаты	Центр	Площадь
1.	Шестиугольник	[13,12][12.5,12.866][11.5,12.866] [11,12] [11.5,11.134] [12.5,11.134]	[12,12]	2
2.	Шестиугольник	[11,9][6,17.6603][-4,17.6603] [-9,9][-4,0.339746][6,0.339746]	[1,9]	259
3.	Восьмиугольник	[3,0][2.12132,2.12132] [1.83697e-16,3][-2.12132,2.12132] [-3,3.67394e-16][-2.12132,-2.12132] [-5.51091e-16,-3][2.12132,-2.12132-]	[0,0]	25

4.	Восьмиугольник	[328,322] [298.711,392.711] [228,422] [157.289,392.711] [128,322] [157.289,251.289][228,222] [298.711,251.289]	[228,322]	28284
5.	Пятиугольник	[103,100][100.927,102.853] [97.5729,101.763] [97.5729,98.2366] [100.927,97.1468]	[100,100]	21
6.	Пятиугольник	[0,3] [2.853, 0.927] [1.763, -2.427] [-1.763,-2.427] [-2.853, 0.927]	[0,0]	21

5. Объяснение результатов работы программы

Программа печатает в консоль меню, в которой описан весь возможный функционал: ввод различных фигур, по координатам, запись и хранение фигур в векторе указателей на фигуры, подсчет центров и площадей фигур, а также суммарной площади. Для решени данного задания было разработано 3 класса: класс вершин, фигур и фигур по заданию, которые наследуются от базового класса Figure, для каждого такого класса были переопределены функции нахождения центра, площади, а также вывод координат, при чем способ вычисления площади фигур находится по разному, в зависимости от типа фигуры.

6. Вывод

С помощью наследования программист может использовать универсальные классы и подстраивать их под себя, добавляя или изменяя функционал субкласса, для этого у программиста есть целый ряд функций и возможностей, например программист может переопределить virtual-методы субкласса так, как того требует задание, использовать данные и информацию уже описанного субкласса и добавлять к нему свои данные и методы.