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, №12 |
| Студент: | Кузьмичев Александр Николаевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 13.11.2019 |

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 12: ромб, трапеция, пятиугольник*

1. **Адрес репозитория на GitHub**

https://github.com/poisoned-monkey/oop\_exercise\_03

1. **Код программы на С++**

*main.cpp*

#include<iostream>  
#include"Figures.h"  
#include<locale>  
  
**int** option() {  
 **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 коориднаты 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::cin >> Menu;  
 **return** Menu;  
}  
  
**int** figure() {  
 **int** Menu;  
 std::cout << "1. Ввести трапецию" << std::endl;  
 std::cout << "2. Ввести ромб" << std::endl;  
 std::cout << "3. Ввести пятиугольник" << 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 = option()) {  
 **case** 1:  
 **switch** (Menu\_2 = figure()) {  
 **case** 1:  
 f = **new** Trapeze{ std::cin };  
 **break**;  
 **case** 2:  
 f = **new** Rhombus{ std::cin };  
 **break**;  
 **case** 3:  
 f = **new** Pentagon(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]).print();  
 std::cout << std::endl;  
 **continue**;  
 **case** 5:  
 **for** (**int** i = 0; i < (**int**)Figures.size(); i++)  
 **if** (Figures[i] != **nullptr**) {  
 (\*Figures[i]).print();  
 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**;  
 **default**:  
 **for** (**int** i = 0; i < (**int**)Figures.size(); i++) {  
 **delete** Figures[i];  
 Figures[i] = **nullptr**;  
 }  
  
 **return** 0;  
 }  
 }  
 **return** 0;  
}

*vertex.h*

#pragma once  
#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.h"  
#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.h*

#pragma once  
#include<iostream>  
#include<vector>  
#include"Vertex.h"  
**class** Figure {  
**public**:  
 **virtual** Vertex center() **const** = 0;  
 **virtual double** square() **const** = 0;  
 **virtual void** print() **const** = 0;  
};

*figs.h*

#include<iostream>  
#include"Fig.h"  
**class** Trapeze : **public** Figure {  
**private**:  
 Vertex Vertexes[4];  
**public**:  
 Trapeze();  
 Trapeze(std::istream& in);  
 Vertex center() **const** override;  
  
 **double** square() **const** override;  
  
 **void** print() **const** override;  
};  
  
**class** Rhombus : **public** Figure {  
**private**:  
 Vertex Vertexes[4];  
**public**:  
 Rhombus();  
 Rhombus(std::istream& in);  
  
 Vertex center() **const** override;  
  
 **double** square() **const** override;  
  
 **void** print() **const** override;  
};  
  
**class** Pentagon : **public** Figure {  
**private**:  
 Vertex Vertexes[5];  
**public**:  
 Pentagon();  
 Pentagon(std::istream& in);  
  
 Vertex center() **const** override;  
  
 **double** square() **const** override;  
  
 **void** print() **const** override;  
};

*figs.cpp*

#include<iostream>  
#include"Figures.h"  
#include<cmath>  
#include<cassert>  
  
Trapeze::Trapeze() {};  
Trapeze::Trapeze(std::istream& in) {  
 in >> Vertexes[0].x >> Vertexes[0].y >> Vertexes[1].x >> Vertexes[1].y >> Vertexes[2].x >> Vertexes[2].y >> Vertexes[3].x >> Vertexes[3].y;  
 assert(vector\_product(Vertexes[0] - Vertexes[3], Vertexes[1] - Vertexes[2]) == 0);  
}  
  
Vertex Trapeze::center() **const** {  
  
 Vertex res;  
 **for** (**int** i = 0; i<4; i++)  
 res += Vertexes[i];  
 **return** res / 4;  
}  
  
**double** Trapeze::square() **const** {  
 **double** Area = 0;  
 **for** (**int** i = 0; i < 4; i++) {  
 Area += (Vertexes[i].x) \* (Vertexes[(i + 1) % 4].y) - (Vertexes[(i + 1) % 4].x)\*(Vertexes[i].y);  
 }  
 Area \*= 0.5;  
 **return** abs(Area);  
}  
  
**void** Trapeze::print() **const** {  
 std::cout << "Trapeze: ";  
 **for** (**int** i = 0; i < 4; i++)  
 std::cout << Vertexes[i] << ' ';  
 std::cout << '\b';  
}  
  
  
  
Rhombus::Rhombus() {};  
Rhombus::Rhombus(std::istream& in) {  
 in >> Vertexes[0].x >> Vertexes[0].y >> Vertexes[1].x >> Vertexes[1].y >> Vertexes[2].x >> Vertexes[2].y >> Vertexes[3].x >> Vertexes[3].y;  
 assert((distance(Vertexes[0], Vertexes[3]) == distance(Vertexes[0], Vertexes[1])) && (distance(Vertexes[0], Vertexes[3]) == distance(Vertexes[1], Vertexes[2])) && (distance(Vertexes[0], Vertexes[3]) == distance(Vertexes[2], Vertexes[3])));  
}  
  
Vertex Rhombus::center() **const** {  
 Vertex res = Vertex();  
 **for** (**int** i = 0; i<4; i++)  
 res += Vertexes[i];  
 **return** res / 4;  
}  
  
**double** Rhombus::square() **const** {  
 **double** Area = 0;  
 **for** (**int** i = 0; i < 4; i++) {  
 Area += (Vertexes[i].x) \* (Vertexes[(i + 1) % 4].y) - (Vertexes[(i + 1) % 4].x)\*(Vertexes[i].y);  
 }  
 Area \*= 0.5;  
 **return** abs(Area);  
}  
  
**void** Rhombus::print() **const** {  
 std::cout << "Rhombus: ";  
 **for** (**int** i = 0; i < 4; i++)  
 std::cout << Vertexes[i] << ' ';  
 std::cout << '\b';  
}  
  
  
  
Pentagon::Pentagon() {};  
Pentagon::Pentagon(std::istream& in) {  
 in >> Vertexes[0].x >> Vertexes[0].y >> Vertexes[1].x >> Vertexes[1].y >> Vertexes[2].x >> Vertexes[2].y >> Vertexes[3].x >> Vertexes[3].y >> Vertexes[4].x >> Vertexes[4].y;  
}  
  
Vertex Pentagon::center() **const** {  
 Vertex res = Vertex();  
 **for** (**int** i = 0; i < 5; i++)  
 res += Vertexes[i];  
 **return** res / 5;  
}  
**double** Pentagon::square() **const** {  
 **double** Area = 0;  
 **for** (**int** i = 0; i < 5; i++) {  
 Area += (Vertexes[i].x) \* (Vertexes[(i + 1)%5].y) - (Vertexes[(i + 1)%5].x)\*(Vertexes[i].y);  
 }  
 Area \*= 0.5;  
 **return** abs(Area);  
}  
  
**void** Pentagon::print() **const** {  
 std::cout << "Pentagon: ";  
 **for** (**int** i = 0; i < 5; i++)  
 std::cout << Vertexes[i] << ' ';  
 std::cout << '\b';  
}

*CMAkeLists.txt*

cmake\_minimum\_required(VERSION 3.12)  
project(laba\_OOP\_03)  
  
set(CMAKE\_CXX\_STANDARD 14)  
  
add\_executable(laba\_OOP\_03 main.cpp Trap.cpp Rhomb.cpp Pent.cpp Point.cpp Figure.cpp Vector.cpp)

1. **Результаты выполнения тестов**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Фигура** | **Координаты** | **Центр** | **Площадь** |
| 1. | Ромб | [1,0] [0,1] [1,2] [2,1] | [1, 1] | 1 |
| 2. | Ромб | [0,4] [3,0] [7,3] [4,7] | [3.5, 3.5] | 12.5 |
| 3. | Трапеция | [0,0] [3,0] [1,1] [2,1] | [1.5,0.5] | 2 |
| 4. | Трапеция | [0,1] [2,2] [2,5] [0,6] | [1,3.5] | 8 |
| 5. | Трапеция | [0,0] [0,2] [2,4] [4,4] | [1.5, 2.5] | 6 |
| 6. | Пятиугольник | [0,3] [2.853, 0.927] [1.763, -2.427] [-1.763,-2.427] [-2.853, 0.927] | [0,0] | 56.0196 |

1. **Объяснение результатов работы программы**

Программа выводит меню, в котором описан весь возможный функционал: ввод фигур: трапеции, ромба и пятиугольника по координатам, запись и хранение фигур в векторе указателей на фигуры, подсчет центров и площадей фигур, а также суммарной площади. Для решения данного задания было реализовано 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. Способ вычисления площади фигур находится по разному, в зависимости от типа фигуры.

1. **Вывод**

В данной лабораторной работе я познакомился с механизмом наследования классов, являющимся одним из основополагающих принципов ООП. Также я научился переопределять virtual-методы классов.