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-108Б-18, №12 |
| Студент: | Коростелев Дмитрий Васильевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 28.10.2019 |

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/Dmitry4K/oop\_exercise\_](https://github.com/Dmitry4K/oop_exercise_2)03

1. **Код программы на С++**

*main.cpp*

#include<iostream>

#include"figs.h"

#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 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 whatFigure() {

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 = getOption()) {

case 1:

switch (Menu\_2 = whatFigure()) {

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]).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;

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*

#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 printCords() const = 0;

//virtual ~Figure();

};

*figs.h*

#pragma once

#include<iostream>

#include"figure.h"

class Trapeze : public Figure {

private:

Vertex Vertexs[4];

public:

Trapeze();

Trapeze(std::istream& in);

Vertex center() const override;

double square() const override;

void printCords() const override;

};

class Rhombus : public Figure {

private:

Vertex Vertexs[4];

public:

Rhombus();

Rhombus(std::istream& in);

Vertex center() const override;

double square() const override;

void printCords() const override;

};

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;

};

*figs.cpp*

#include<iostream>

#include"figs.h"

#include<cmath>

#include<cassert>

//Trapeze

Trapeze::Trapeze() {};

Trapeze::Trapeze(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;

assert(vector\_product(Vertexs[0] - Vertexs[3], Vertexs[1] - Vertexs[2]) == 0);

}

Vertex Trapeze::center() const {

Vertex res;

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

res += Vertexs[i];

return res / 4;

}

double Trapeze::square() const {

double squareSum = (Vertexs[2].y - Vertexs[0].y)\*(Vertexs[3].x - Vertexs[0].x);

double squareMin = (Vertexs[1].x - Vertexs[0].x)\*(Vertexs[2].y - Vertexs[2].y);

double squareBottom = (Vertexs[0].y \* (Vertexs[3].x - Vertexs[0].x));

double triangleUpLeft = (Vertexs[2].y - Vertexs[1].y)\*(Vertexs[2].x - Vertexs[1].x)\*0.5;

double triangleUpRight = (Vertexs[2].y - Vertexs[3].y)\*(Vertexs[3].x - Vertexs[2].x) \* 0.5;

double triangleRightBottom = (Vertexs[3].y - Vertexs[0].y)\*(Vertexs[3].x - Vertexs[0].x) \* 0.5;

double triangleLeftBottom = (Vertexs[1].y - Vertexs[0].y)\*(Vertexs[1].x - Vertexs[0].x)\*0.5;

return squareSum - squareMin - squareBottom - triangleUpLeft - triangleUpRight - triangleRightBottom - triangleLeftBottom;

/\*double a, b, c;

if (vector\_product(Vertexs[0] - Vertexs[1], Vertexs[2] - Vertexs[3]) == 0) {

a = distance(Vertexs[0], Vertexs[1]);

b = distance(Vertexs[2], Vertexs[3]);

c = distance(Vertexs[0], Vertexs[2]) < distance(Vertexs[0], Vertexs[3]) ? distance(Vertexs[0], Vertexs[2]) : distance(Vertexs[0], Vertexs[3]);

}

else if (vector\_product(Vertexs[0] - Vertexs[3], Vertexs[1] - Vertexs[2]) == 0) {

a = distance(Vertexs[0], Vertexs[3]);

b = distance(Vertexs[1], Vertexs[2]);

c = distance(Vertexs[0], Vertexs[1]) < distance(Vertexs[0], Vertexs[2]) ? distance(Vertexs[0], Vertexs[1]) : distance(Vertexs[0], Vertexs[2]);

}

else {

a = distance(Vertexs[1], Vertexs[3]);

b = distance(Vertexs[0], Vertexs[2]);

c = distance(Vertexs[1], Vertexs[0]) < distance(Vertexs[1], Vertexs[2]) ? distance(Vertexs[1], Vertexs[0]) : distance(Vertexs[1], Vertexs[2]);

}

return ((a + b) / 2)\*sqrt(pow(c, 2) - pow(((b - a) / 2), 2));\*/

}

void Trapeze::printCords() const {

std::cout << "Trapeze: ";

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

std::cout << Vertexs[i] << ' ';

std::cout << '\b';

}

//Rhombus

Rhombus::Rhombus() {};

Rhombus::Rhombus(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;

assert((distance(Vertexs[0], Vertexs[3]) == distance(Vertexs[0], Vertexs[1])) && (distance(Vertexs[0], Vertexs[3]) == distance(Vertexs[1], Vertexs[2])) && (distance(Vertexs[0], Vertexs[3]) == distance(Vertexs[2], Vertexs[3])));

}

Vertex Rhombus::center() const {

Vertex res = Vertex();

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

res += Vertexs[i];

return res / 4;

}

double Rhombus::square() const {

double squareAll = (Vertexs[2].x - Vertexs[0].x)\*(Vertexs[2].y - Vertexs[0].y); //2-0

double squareLeft = (Vertexs[1].x - Vertexs[0].x)\*(Vertexs[2].y - Vertexs[1].y); // 1-0 2-1

double squareRight = (Vertexs[2].x - Vertexs[3].x)\*(Vertexs[3].y - Vertexs[0].y); //2-3 3-0

double squareBottom = (Vertexs[2].x - Vertexs[0].x)\*(Vertexs[0].y);//

double triangleUp = 0.5 \* (Vertexs[2].x - Vertexs[1].x)\*(Vertexs[2].y - Vertexs[1].y);//2-1 2-1

double triangleLeft = 0.5 \* (Vertexs[1].x - Vertexs[0].x)\*(Vertexs[1].y - Vertexs[0].y);//1-0 1-0

double triangleRight = 0.5 \* (Vertexs[2].x - Vertexs[3].x)\*(Vertexs[2].y - Vertexs[3].y);//2-3 2-3

double triangleBottom = 0.5 \*(Vertexs[3].x - Vertexs[0].x)\*(Vertexs[3].y - Vertexs[0].y);//3-0 3-0

return squareAll - squareLeft - squareRight - squareBottom - triangleUp - triangleLeft - triangleRight - triangleBottom;

//return distance(Vertexs[0], Vertexs[2])\*distance(Vertexs[1], Vertexs[3]) / 2;

}

void Rhombus::printCords() const {

std::cout << "Rhombus: ";

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

std::cout << Vertexs[i] << ' ';

std::cout << '\b';

}

//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';

}

*CMAkeLists.txt*

cmake\_minimum\_required(VERSION 3.0)

project(lab3)

set(SOURCE\_EXE main.cpp)

set(SOURCE\_LIB\_FIGURES figs.cpp)

set(SOURCE\_LIB\_VERTEX vertex.cpp)

add\_library(figures STATIC ${SOURCE\_LIB\_FIGURES})

add\_library(vertex STATIC ${SOURCE\_LIB\_VERTEX})

add\_executable(main ${SOURCE\_EXE})

target\_link\_libraries(main figures figure vertex)

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 печатает в консоль меню, в которой описан весь возможный функционал: ввод различных 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 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.

1. **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.