**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: Мариничев Иван Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата: 26.10.20

Оценка:

Москва, 2020

1. Постановка задачи. Вариант 31

Разработать классы **6-угольник**, **8-угольник** и **треугольник**, классы должны наследоваться от базового класса **Figure**. Фигуры являются выпуклыми равносторонними 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.

1. Описание программы

Программа предназначена для работы с тремя типами равносторонних фигур: шестиугольник, восьмиугольник и треугольник. Программа получает на ввод координаты вершин соответствующей фигуры (по часовой стрелке), при вводе фигуры происходит проверка фигуры на правильность. Программа может вычислить площадь данной фигуры, геометрический центр, а также вывести координаты вершин фигуры. Также пользователь может добавить фигуру в динамический массив по индексу или удалить ее, для которого тоже можно вычислить общую площадь.

Программа имеет пользовательский интерфейс:

1) Добавить фигуру;

2) Удалить фигуру;

3) Вычислить геометрический центр;

4) Найти координаты фигуры;

5) Найти площадь фигуры;

6) Найти общую площадь;

7) Выйти;

Пользователь вводит цифру соответствующую пункту меню.

* При добавлении фигуры нужно ввести тип фигуры и координаты
* При удалении нужно ввести индекс фигуры в массиве
* При вычислении геом. центра нужно ввести индекс фигуры в массиве
* При нахождении площади фигуры нужно ввести индекс фигуры в массиве

1. Руководство по использованию программы

Таблица 1 – Функции файла main.cpp

|  |  |  |
| --- | --- | --- |
| Название | Аргументы | Описание |
| void Menu() |  | Вывод пунктов меню |
| int main() | std::vector<Figure \*> figures;  unsigned long long totalArea = 0;  bool input = true;  int option, figureType;  unsigned int index; | Основная функция с помощью которой происходит демонстрация всех возможностей программы |

Таблица 2 – Методы, функции и структуры файла vertex.hpp

|  |  |  |
| --- | --- | --- |
| Название | Аргументы | Описание |
| Class Vertex | double x;  double y;  double distance(Vertex &v) const; | Координата x  Координата y  Расстояние между двумя вершинами |
| Vertex() |  | Конструктор по умолчанию |
| Vertex() | double x, double y | Конструктор |
| ~Vertex() |  | Деструктор |
| ostream &operator<<(); | ostream &out, Vertex vertex | Перегрузка оператора вывода |
| istream &operator>>(); | istream &in, Vertex &vertex | Перегрузка оператора ввода |

Таблица 3 – Методы, функции и структуры файла figure.hpp

|  |  |  |
| --- | --- | --- |
| Название | Аргументы | Описание |
| class Figure | std::vector<Vertex> vertices; //Вектор вершин  std::string type; //Тип фигуры | Базовый класс |
| virtual Vertex CalculateGeometricMedian() |  | Метод вычисления геометрического центра фигуры |
| virtual void DisplayFigureCoordinates() |  | Метод вывода координат вершин фигуры |
| friend std::istream &operator>>() | std::istream &in, Figure &figure | Перегрузка оператора ввода |
| virtual ~Figure() |  | Деструктор |
| bool check\_vertices() |  | Проверка координат вершин на соответствие условию правильности фигуры |

Таблица 4 – Методы, функции и структуры файла hexagon.hpp

|  |  |  |
| --- | --- | --- |
| Название | Аргументы | Описание |
| class Hexagon |  | Класс шестиугольник |
| Hexagon(); |  | Конструктор |
| ~Hexagon(); |  | Деструктор |
| Vertex CalculateGeometricMedian() override; |  | Метод вычисления геометрического центра шестиугольника |
| void DisplayFigureCoordinates() override; |  | Метод вывода координат вершин шестиугольника |
| double CalculateFigureArea() override; |  | Метод вычисления площади шестиугольника |

Таблица 5 – Методы, функции и структуры файла octagon.hpp

|  |  |  |
| --- | --- | --- |
| Название | Аргументы | Описание |
| class Octagon |  | Класс восьмиугольник |
| Octagon(); |  | Конструктор |
| ~Octagon(); |  | Деструктор |
| Vertex CalculateGeometricMedian() override; |  | Метод вычисления геометрического центра восьмиугольника |
| void DisplayFigureCoordinates() override; |  | Метод вывода координат вершин восьмиугольника |
| double CalculateFigureArea() override; |  | Метод вычисления площади восьмиугольника |

Таблица 6 – Методы, функции и структуры файла triangle.hpp

|  |  |  |
| --- | --- | --- |
| Название | Аргументы | Описание |
| class Triangle |  | Класс треугольник |
| Triangle(); |  |  |
| ~Triangle(); |  |  |
| Vertex CalculateGeometricMedian() override; |  | Метод вычисления геометрического центра треугольника |
| void DisplayFigureCoordinates() override; |  | Метод вывода координат вершин треугольника |
| double CalculateFigureArea() override; |  | Метод вычисления площади треугольника |

**Результаты выполнения тестов**

|  |  |  |
| --- | --- | --- |
| Название тестового файла | Входные данные | Результат |
| test\_01.txt | 1  3  0 0  0.5 0.86  1 0  1  1  0.866025403 -0.5  0.866025403 0.5  0 1  -0.866025403 0.5  -0.866025403 -0.5  0 -1  3  0  3  1  4  0  4  1  5  0  5  1  6  7 | 1) Add figure  2) Remove figure  3) Calculate geometric median of the figure  4) Find coordinates of figure's vertices  5) Find area of figure  6) Find total area  7) Exit  1  Enter the type of figure (1 - hexagon, 2 - octagon, 3 - triangle)  3  Enter vertices of this figure  0 0  0.5 0.86  1 0  Figure succesfully added at index 0  1  Enter the type of figure (1 - hexagon, 2 - octagon, 3 - triangle)  1  Enter vertices of this figure  0.866025403 -0.5  0.866025403 0.5  0 1  -0.866025403 0.5  -0.866025403 -0.5  0 -1  Figure succesfully added at index 1  3  Enter index of the figure:  0  Figure's geometric median: (0.5;0.287)  3  Enter index of the figure:  1  Figure's geometric median: (0;0)  4  Enter index of the figure:  0  Coordinates of Figure's vertices: (0;0), (0.5;0.86), (1;0)  4  Enter index of the figure:  1  Coordinates of Figure's vertices: (0.866;-0.5), (0.866;0.5), (0;1), (-0.866;0.5), (-0.866;-0.5), (0;-1)  5  Enter index of the figure:  0  Figure's area: 0.429  6  Enter index of the figure:  0  Coordinates of Figure's vertices: (0;0), (0.5;0.86), (1;0)  4  Enter index of the figure:  1  Coordinates of Figure's vertices: (0.866;-0.5), (0.866;0.5), (0;1), (-0.866;0.5), (-0.866;-0.5), (0;-1)  5  Enter index of the figure:  0  Figure's area: 0.429  6  Total area: 2  5  Enter index of the figure:  1  Figure's area: 2.6  6  Total area: 2  7 |
| test\_02.txt | 1  1  0.866025403 -0.5  0.866025403 0.5  0 1  -0.866025403 0.5  -0.866025403 -0.5  0 -1  1  3  0 0  0.86 0.5  0 1  3  0  3  1  4  0  4  1  5  0  5  1  6  7 | 1) Add figure  2) Remove figure  3) Calculate geometric median of the figure  4) Find coordinates of figure's vertices  5) Find area of figure  6) Find total area  7) Exit  1  Enter the type of figure (1 - hexagon, 2 - octagon, 3 - triangle)  1  Enter vertices of this figure  0.866025403 -0.5  0.866025403 0.5  0 1  -0.866025403 0.5  -0.866025403 -0.5  0 -1  Figure succesfully added at index 0  1  Enter the type of figure (1 - hexagon, 2 - octagon, 3 - triangle)  1  Enter vertices of this figure  0.866025403 -0.5  0.866025403 0.5  0 1  -0.866025403 0.5  -0.866025403 -0.5  0 -1  Figure succesfully added at index 1  3  Enter index of the figure:  0  Figure's geometric median: (0;0)  3  Enter index of the figure:  1  Figure's geometric median: (0;0)  4  Enter index of the figure:  0  Coordinates of Figure's vertices: (0.866;-0.5), (0.866;0.5), (0;1), (-0.866;0.5), (-0.866;-0.5), (0;-1)  4  Enter index of the figure:  1  Coordinates of Figure's vertices: (0.866;-0.5), (0.866;0.5), (0;1), (-0.866;0.5), (-0.866;-0.5), (0;-1)  5  Enter index of the figure:  0  Figure's area: 2.6  5  Enter index of the figure:  1  Figure's area: 2.6  6  Total area: 4  7 |

1. Листинг программы

**main.cpp**

/\* Мариничев Иван М8О-208Б-19

\*

\* github: IvaMarin

\*

\* Вариант 31:

\* Разработать классы 6-угольник, 8-угольник и треугольник, классы должны наследоваться от базового класса Figure.

\* Фигуры являются выпуклыми равносторонними 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.

\*/

#include <iostream>

#include <vector>

#include <string>

#include "figure.hpp"

#include "hexagon.hpp"

#include "octagon.hpp"

#include "triangle.hpp"

using namespace std;

void Menu()

{

cout << "1) Add figure\n";

cout << "2) Remove figure\n";

cout << "3) Calculate geometric median of the figure\n";

cout << "4) Find coordinates of figure's vertices\n";

cout << "5) Find area of figure\n";

cout << "6) Find total area\n";

cout << "7) Exit\n";

}

int main() {

Menu();

std::vector<Figure \*> figures;

unsigned long long totalArea = 0;

bool input = true;

int option, figureType;

unsigned int index;

while (input) {

cin >> option;

switch (option) {

case 1: {

cout << "Enter the type of figure (1 - hexagon, 2 - octagon, 3 - triangle)" << endl;

cin >> figureType;

if (figureType == 1) {

auto \*H = new Hexagon;

cout << "Enter vertices of this figure" << endl;

cin >> \*H;

figures.push\_back(H);

} else if (figureType == 2) {

auto \*O = new Octagon;

cout << "Enter vertices of this figure" << endl;

cin >> \*O;

figures.push\_back(O);

} else if (figureType == 3) {

auto \*T = new Triangle;

cout << "Enter vertices of this figure" << endl;

cin >> \*T;

figures.push\_back(T);

}

totalArea += figures.back()->CalculateFigureArea();

cout << "Figure succesfully added at index " << figures.size() - 1 << '\n';

break;

}

case 2: {

cout << "Enter index of figure you want to delete: " << endl;

cin >> index;

if (index > figures.size()) {

cout << "Index is out of reach!\n";

}

else {

totalArea -= figures[index]->CalculateFigureArea();

cout << "Figure successfully removed\n";

delete figures[index];

figures.erase(figures.begin() + index);

}

break;

}

case 3: {

cout << "Enter index of the figure: " << endl;

cin >> index;

if (index > figures.size()) {

cout << "Index is out of reach!\n";

}

else {

cout << "Figure's geometric median: " << figures[index]->CalculateGeometricMedian() << '\n';

}

break;

}

case 4: {

cout << "Enter index of the figure: " << endl;

cin >> index;

if (index > figures.size()) {

cout << "Index is out of reach!\n";

}

else {

cout << "Coordinates of " << "Figure's vertices:\t";

figures[index]->DisplayFigureCoordinates();

}

break;

}

case 5: {

cout << "Enter index of the figure: " << endl;

cin >> index;

if (index > figures.size()) {

cout << "Index is out of reach!\n";

}

else {

cout << "Figure's area: " << figures[index]->CalculateFigureArea() << '\n';

}

break;

}

case 6: {

cout << "Total area: " << totalArea << endl;

break;

}

case 7: {

input = false;

break;

}

}

}

for(auto &f : figures) {

delete f;

}

return 0;

}

**vertex.hpp**

#ifndef VERTEX\_HPP

#define VERTEX\_HPP

#include <iostream>

#include <cmath>

using namespace std;

class Vertex { //Класс вершина

public:

double x; //Координата x

double y; //Координата y

double distance(Vertex &v) const; //Расстояние между двумя вершинами

Vertex(); //Конструктор по умолчанию

Vertex(double x, double y); //Конструктор

~Vertex(); //Деструктор

};

ostream &operator<<(ostream &out, Vertex vertex); //Перегрузка оператора вывода

istream &operator>>(istream &in, Vertex &vertex); //Перегрузка оператора ввода

#endif /\* VERTEX\_HPP \*/

**vertex.cpp**

#include "vertex.hpp"

double Vertex::distance(Vertex &v) const {

return sqrt(pow(v.x - x, 2) + pow(v.y - y, 2));

}

Vertex::Vertex(): x(0.0), y(0.0) {}

Vertex::Vertex(double x, double y) : x(x), y(y) {}

Vertex::~Vertex() {}

ostream &operator<<(ostream &out, Vertex vertex) {

cout.precision(3);

out << "(" << vertex.x << ";" << vertex.y << ")";

return out;

}

istream &operator>>(istream &in, Vertex &vertex) {

in >> vertex.x >> vertex.y;

return in;

}

**figure.hpp**

#ifndef FIGURE\_HPP

#define FIGURE\_HPP

#include <vector>

#include "vertex.hpp"

class Figure { //Базовый класс

public:

virtual Vertex CalculateGeometricMedian() = 0; //Метод вычисления геометрического центра фигуры

virtual void DisplayFigureCoordinates() = 0; //Метод вывода координат вершин фигуры

virtual double CalculateFigureArea() = 0; //Метод вычисления площади фигуры

friend std::istream &operator>>(std::istream &in, Figure &figure) {

do {

for (auto &vertex : figure.vertices) {

in >> vertex;

}

} while (!figure.check\_vertices());

return in;

}

virtual ~Figure() {}

protected:

std::vector<Vertex> vertices; //Вектор вершин

std::string type;

bool check\_vertices() {

double figure\_length = vertices[0].distance(vertices[vertices.size() - 1]);

for (unsigned int i = 0; i < vertices.size() - 1; ++i) {

double cur\_length = vertices[i].distance(vertices[i + 1]);

if (std::abs(cur\_length - figure\_length) >= 1e-2) {

std::cout << "Figure must have equal sides. Try again!" << std::endl;

return false;

}

if (figure\_length == 0) {

std::cout << "Points should be different. Try again!" << std::endl;

return false;

}

}

return true;

}

};

#endif /\* FIGURE\_HPP \*/

**hexagon.hpp**

#ifndef HEXAGON\_HPP

#define HEXAGON\_HPP

#include "figure.hpp"

class Hexagon: public Figure { //Класс шестиугольник

public:

Hexagon(); //Конструктор

~Hexagon(); //Деструктор

Vertex CalculateGeometricMedian() override; //Метод вычисления геометрического центра шестиугольника

void DisplayFigureCoordinates() override; //Метод вывода координат вершин шестиугольника

double CalculateFigureArea() override; //Метод вычисления площади шестиугольника

};

#endif /\* HEXAGON\_HPP \*/

**hexagon.cpp**

#include "hexagon.hpp"

Hexagon::Hexagon() {

vertices.resize(6);

type = "Hexagon";

}

Hexagon::~Hexagon() {}

Vertex Hexagon::CalculateGeometricMedian() {

double sum\_x = 0, sum\_y = 0;

for (Vertex &vertex : vertices) {

sum\_x += vertex.x;

sum\_y += vertex.y;

}

return Vertex(sum\_x / 6.0, sum\_y / 6.0);

}

void Hexagon::DisplayFigureCoordinates() {

for (int i = 0; i < 6; ++i) {

cout << vertices[i];

if(i != 5) cout << ", ";

}

cout << '\n';

}

double Hexagon::CalculateFigureArea() {

double side = vertices[0].distance(vertices[1]);

return pow(side, 2) \* 3 \* sqrt(3) / 2;

}

**octagon.hpp**

#ifndef OCTAGON\_HPP

#define OCTAGON\_HPP

#include "figure.hpp"

class Octagon: public Figure { //Класс восьмиугольник

public:

Octagon(); //Конструктор

~Octagon(); //Деструктор

Vertex CalculateGeometricMedian() override; //Метод вычисления геометрического центра восьмиугольника

void DisplayFigureCoordinates() override; //Метод вывода координат вершин восьмиугольника

double CalculateFigureArea() override; //Метод вычисления площади восьмиугольника

};

#endif /\* OCTAGON\_HPP \*/

**octagon.cpp**

#include "octagon.hpp"

Octagon::Octagon() {

vertices.resize(8);

type = "Octagon";

}

Octagon::~Octagon() {}

Vertex Octagon::CalculateGeometricMedian() {

double sum\_x = 0, sum\_y = 0;

for (Vertex &vertex : vertices) {

sum\_x += vertex.x;

sum\_y += vertex.y;

}

return Vertex(sum\_x / 6.0, sum\_y / 6.0);

}

void Octagon::DisplayFigureCoordinates() {

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

cout << vertices[i];

if(i != 7) cout << ", ";

}

cout << '\n';

}

double Octagon::CalculateFigureArea() {

double side = vertices[0].distance(vertices[1]);

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

}

**triangle.hpp**

#ifndef TRIANGLE\_HPP

#define TRIANGLE\_HPP

#include "figure.hpp"

class Triangle: public Figure { //Класс треугольник

public:

Triangle(); //Конструктор

~Triangle(); //Деструктор

Vertex CalculateGeometricMedian() override; //Метод вычисления геометрического центра треугольника

void DisplayFigureCoordinates() override; //Метод вывода координат вершин треугольника

double CalculateFigureArea() override; //Метод вычисления площади треугольника

};

#endif /\* TRIANGLE\_HPP \*/

**triangle.cpp**

#include "triangle.hpp"

Triangle::Triangle() {

vertices.resize(3);

type = "Triangle";

}

Triangle::~Triangle() {}

Vertex Triangle::CalculateGeometricMedian() {

double sum\_x = 0, sum\_y = 0;

for (Vertex &vertex : vertices) {

sum\_x += vertex.x;

sum\_y += vertex.y;

}

return Vertex(sum\_x / 3.0, sum\_y / 3.0);

}

void Triangle::DisplayFigureCoordinates() {

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

cout << vertices[i];

if(i != 2) cout << ", ";

}

cout << '\n';

}

double Triangle::CalculateFigureArea() {

double side = vertices[0].distance(vertices[1]);

return (pow(side, 2) \* sqrt(3) ) / 4;

}

**CmakeLists.txt**

cmake\_minimum\_required(VERSION 3.16.3)

set(CMAKE\_CXX\_FLAGS "-pedantic -Wall -Wextra -Werror")

set(CMAKE\_CXX\_STANDART 17)

project(oop\_lab3)

add\_executable(oop\_exercise\_03 vertex.cpp hexagon.cpp octagon.cpp triangle.cpp main.cpp)

1. Вывод

Список литературы

1. Стефан К. Дьюхэрст Скользкие места С++. Как избежать проблем при проектировании и компиляции ваших программ — ISBN: 5-94074-083-9 — 266 с.
2. Inheritance in C++ [Электронный ресурс]. URL: https://www.geeksforgeeks.org/inheritance-in-c/ (дата обращения: 18.10.2020).
3. Geometric Median [Электронный ресурс]. URL: https://www.geeksforgeeks.org/geometric-median/ (дата обращения: 19.10.2020).
4. Площадь правильного шестиугольника [Электронный ресурс]. URL: https://www.fxyz.ru/формулы\_по\_геометрии/формулы\_площади/площадь\_правильного\_шестиугольника/ (дата обращения: 22.10.2020).
5. Площадь правильного восьмиугольника [Электронный ресурс]. URL: https://www.rapidus.ru/area-8angle.html (дата обращения: 23.10.2020).