**30333-2 Калевич Сергей Михайлович Лабораторная работа №7, вариант 4**

**Оглавление**

[ЗАДАНИЕ 1. 2](#_Toc159548187)

[Ход выполнения: 2](#_Toc159548188)

[Результат: 16](#_Toc159548189)

[Создание структур. 16](#_Toc159548190)

[Вывод информации о фигуре на экран. 18](#_Toc159548191)

[Сохранение структуры в бинарный файл. 19](#_Toc159548192)

[Чтение структуры из файла. 21](#_Toc159548193)

[Выводы 25](#_Toc159548194)

ЗАДАНИЕ 1

Разработать структуры для различных геометрических фигур:

* окружность;
* куб.

Разработать перегруженные функции:

* расчет периметра (длина всех ребер, возможен 0);
* расчет площади (площади поверхности);
* расчет объема (возможен 0);
* вывод информации о фигуре на экран;
* сохранение структуры в бинарный файл (имя файла как параметр);
* чтение структуры из файла (имя файла и указатель на структуру – параметры функции).

Ход выполнения

// Lab #7

// task #1

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <cmath>

#include <Windows.h>

using namespace std;

struct Circle

{

int x{}, y{}, radius{};

bool isDefined{ false };

} circle;

struct Cube

{

double vertexes[8][3] =

{

{NULL, NULL, NULL}, // A

{NULL, NULL, NULL}, // B

{NULL, NULL, NULL}, // C

{NULL, NULL, NULL}, // D

{NULL, NULL, NULL}, // E

{NULL, NULL, NULL}, // F

{NULL, NULL, NULL}, // G

{NULL, NULL, NULL} // H

};

double edgeLength{};

bool isDefined{ false };

} cube;

void drawCube()

{

printf("%25s\n", " H-------------G");

printf("%25s\n", " /' /|");

printf("%25s\n", " / ' / |");

printf("%25s\n", " / ' / |");

printf("%25s\n", " E---+---------F |");

printf("%25s\n", " | ' | |");

printf("%25s\n", " | D - - - | - C");

printf("%25s\n", " | ' | / ");

printf("%25s\n", " | ' | / ");

printf("%25s\n", " |' |/ ");

printf("%25s\n\n", "A-------------B ");

}

double absEpsilon{ 1e-12 };

double relEpsilon{ 1e-8 };

// Функция для проверки равенства чисел сплавающей точкой

bool approximatelyEqualAbsRel(double a, double b, double absEpsilon, double relEpsilon)

{

double diff = fabs(a - b);

if (diff <= absEpsilon)

return true;

return diff <= ((fabs(a) < fabs(b) ? fabs(b) : fabs(a)) \* relEpsilon);

}

// Проверка является ли создаваемая фигура окружностью

bool isFigure(const Circle& circle)

{

if (circle.radius > 0) return true;

cout << "\nОкружность не создана: "

<< "значение радиуса должно быть больше нуля" << endl;

system("pause");

return false;

}

double twoPointsDistance(

double x1, double y1, double z1,

double x2, double y2, double z2)

{

return sqrt(

pow(x2 - x1, 2) +

pow(y2 - y1, 2) +

pow(z2 - z1, 2)

);

}

// Проверка все ли ребра куба равны

bool isCubeEdgesEqual(const Cube& cube)

{

double tempCubeEdgeLength{};

int count{};

int offset{1};

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

{

// Ребра AB, BC, CD, EF, FG, GH

tempCubeEdgeLength = twoPointsDistance(

cube.vertexes[i][0],

cube.vertexes[i][1],

cube.vertexes[i][2],

cube.vertexes[i + offset][0],

cube.vertexes[i + offset][1],

cube.vertexes[i + offset][2]

);

if (!approximatelyEqualAbsRel(

tempCubeEdgeLength,

cube.edgeLength,

absEpsilon,

relEpsilon)

)

{

return false;

}

if (count < 2)

{

count++;

continue;

}

i++;

// Ребра DA, HE

offset = -3;

tempCubeEdgeLength = twoPointsDistance(

cube.vertexes[i][0],

cube.vertexes[i][1],

cube.vertexes[i][2],

cube.vertexes[i + offset][0],

cube.vertexes[i + offset][1],

cube.vertexes[i + offset][2]

);

if (!approximatelyEqualAbsRel(

tempCubeEdgeLength,

cube.edgeLength,

absEpsilon,

relEpsilon)

)

{

return false;

}

count = 0;

offset = 1;

}

// Ребра AE, BF, CG, DH

offset = 4;

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

{

tempCubeEdgeLength = twoPointsDistance(

cube.vertexes[i][0],

cube.vertexes[i][1],

cube.vertexes[i][2],

cube.vertexes[i + offset][0],

cube.vertexes[i + offset][1],

cube.vertexes[i + offset][2]

);

if (!approximatelyEqualAbsRel(

tempCubeEdgeLength,

cube.edgeLength,

absEpsilon,

relEpsilon)

)

{

return false;

}

}

return true;

}

// Проверка все ли грани куба являются квадратами

bool isAllFacesSquares(const Cube& cube)

{

double isoscelesRightTriangleHypotenuse = sqrt(

2 \* pow(cube.edgeLength, 2)

);

double tempFaceDiagonal{};

int offset{5};

// Диагонали AF, BG, CH

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

{

tempFaceDiagonal = twoPointsDistance(

cube.vertexes[i][0],

cube.vertexes[i][1],

cube.vertexes[i][2],

cube.vertexes[i + offset][0],

cube.vertexes[i + offset][1],

cube.vertexes[i + offset][2]

);

if (!approximatelyEqualAbsRel(

tempFaceDiagonal,

isoscelesRightTriangleHypotenuse,

absEpsilon,

relEpsilon)

)

{

return false;

}

}

// Диагональ DE

tempFaceDiagonal = twoPointsDistance(

cube.vertexes[3][0],

cube.vertexes[3][1],

cube.vertexes[3][2],

cube.vertexes[4][0],

cube.vertexes[4][1],

cube.vertexes[4][2]

);

if (!approximatelyEqualAbsRel(

tempFaceDiagonal,

isoscelesRightTriangleHypotenuse,

absEpsilon,

relEpsilon)

)

{

return false;

}

// Диагонали AC, EG

offset = 2;

for (int i = 0; i < 5; i +=4)

{

tempFaceDiagonal = twoPointsDistance(

cube.vertexes[i][0],

cube.vertexes[i][1],

cube.vertexes[i][2],

cube.vertexes[i + offset][0],

cube.vertexes[i + offset][1],

cube.vertexes[i + offset][2]

);

if (!approximatelyEqualAbsRel(

tempFaceDiagonal,

isoscelesRightTriangleHypotenuse,

absEpsilon,

relEpsilon)

)

{

return false;

}

}

return true;

}

// Проверка является ли создаваемая фигура кубом

bool isFigure(const Cube& cube)

{

if (!isCubeEdgesEqual(cube))

{

cout << "\nКуб не создан: " << "все ребра должны быть равны" << endl;

system("pause");

return false;

}

if (!isAllFacesSquares(cube))

{

cout << "\nКуб не создан: " << "все грани должны быть квадратами" << endl;

system("pause");

return false;

}

return true;

}

void createFigure(Circle& circle)

{

circle.isDefined = false;

cout << "Введите координаты центра окружности (x y): ";

cin >> circle.x >> circle.y;

cout << "Введите радиус: ";

cin >> circle.radius;

if (isFigure(circle))

{

circle.isDefined = true;

cout << "Окружность создана" << endl;

system("pause");

}

}

void createFigure(Cube& cube)

{

cube.isDefined = false;

char vertex{ 'A' };

drawCube();

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

{

cout << "Введите координаты вершины " << vertex++ << " (x y z): ";

cin >> cube.vertexes[i][0] >> cube.vertexes[i][1] >> cube.vertexes[i][2];

}

// Ребро AB

cube.edgeLength = twoPointsDistance(

cube.vertexes[0][0],

cube.vertexes[0][1],

cube.vertexes[0][2],

cube.vertexes[1][0],

cube.vertexes[1][1],

cube.vertexes[1][2]

);

if (isFigure(cube))

{

cube.isDefined = true;

cout << "Куб создан" << endl;

system("pause");

}

}

double perimeter(const Circle& circle)

{

return 2 \* M\_PI \* circle.radius;

}

double perimeter(const Cube& cube)

{

return cube.edgeLength \* 12;

}

double square(const Circle& circle)

{

return M\_PI \* pow(circle.radius, 2);

}

double square(const Cube& cube)

{

return pow(cube.edgeLength, 2) \* 6;

}

double volume(const Cube& cube)

{

return pow(cube.edgeLength, 3);

}

void info(const Circle& circle)

{

if (!circle.isDefined)

{

cout << "Окружность неопределена" << endl;

cout << "Создайте окружность" << endl;

system("pause");

return;

}

cout << "Координаты центра окружности: "

<< '(' << circle.x << ", " << circle.y << ')' << endl;

cout << "Радиус окружности: " << circle.radius << endl;

cout << "Длина окружности: " << perimeter(circle) << endl;

cout << "Площадь круга: " << square(circle) << endl;

system("pause");

}

void info(const Cube& cube)

{

if (!cube.isDefined)

{

cout << "Куб неопределен" << endl;

cout << "Создайте куб" << endl;

system("pause");

return;

}

drawCube();

char vertex{ 'A' };

cout << "Координаты вершин куба: " << endl;

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

{

cout << "Координаты вершины " << vertex++ << ": ";

cout << '(' << cube.vertexes[i][0] << ", "

<< cube.vertexes[i][1] << ", "

<< cube.vertexes[i][2] << ')';

cout << endl;

}

cout << "Длина ребра куба: " << cube.edgeLength << endl;

cout << "Периметр ребер куба: " << perimeter(cube) << endl;

cout << "Площадь куба: " << square(cube) << endl;

cout << "Объем куба: " << volume(cube) << endl;

system("pause");

}

void showInfo()

{

while (true)

{

system("cls");

printf("1. Получить информацию об окружности\n");

printf("2. Получить информацию о кубе\n");

printf("0. В главное меню\n");

char menu[80];

cin >> menu;

switch (menu[0])

{

case '1': info(circle); break;

case '2': info(cube); break;

case '0': return;

default:

printf("Неправильный пункт меню\n");

system("pause");

}

}

}

void setFigure()

{

while (true)

{

system("cls");

printf("1. Создать окружность\n");

printf("2. Создать куб\n");

printf("0. В главное меню\n");

char menu[80];

cin >> menu;

switch (menu[0])

{

case '1': createFigure(circle); break;

case '2': createFigure(cube); break;

case '0': return;

default:

printf("Неправильный пункт меню\n");

system("pause");

}

}

}

void saveCircle(char\* filename)

{

char\* b{ nullptr };

FILE\* file;

if (fopen\_s(&file, filename, "wb"))

{

cout << "Ошибка открытия файла: " << filename << endl;

system("pause");

return;

}

b = (char\*)&circle;

for (int i = 0; i < sizeof(circle); i++)

{

fputc(\*(b++), file);

}

printf("Данные сохранены\n");

if (fclose(file))

{

cout << "Ошибка закрытия файла: " << filename << endl;

}

system("pause");

}

void saveCube(char\* filename)

{

char\* b{ nullptr };

FILE\* file;

if (fopen\_s(&file, filename, "wb"))

{

cout << "Ошибка открытия файла: " << filename << endl;

system("pause");

return;

}

b = (char\*)&cube;

for (int i = 0; i < sizeof(cube); i++)

{

fputc(\*(b++), file);

}

printf("Данные сохранены\n");

if (fclose(file))

{

cout << "Ошибка закрытия файла: " << filename << endl;

}

system("pause");

}

void save()

{

char filename[100];

while (true)

{

system("cls");

printf("1. Сохранить окружность\n");

printf("2. Сохранить куб\n");

printf("0. В главное меню\n\n");

char menu[80];

cin >> menu;

switch (menu[0])

{

case '1':

if (circle.isDefined)

{

printf("Выбранная фигура: окружность\n");

printf("Введите имя файла для сохранения (без .bin): ");

cin >> filename;

strcat\_s(filename, 100, ".bin");

saveCircle(filename);

break;

}

printf("Структура окружности не вводилась\n");

system("pause");

break;

case '2':

if (cube.isDefined)

{

printf("Выбранная фигура: куб\n");

printf("Введите имя файла для сохранения (без .bin): ");

cin >> filename;

strcat\_s(filename, 100, ".bin");

saveCube(filename);

break;

}

printf("Структура куба не вводилась\n");

system("pause");

break;

case '0': return;

default:

printf("Неправильный пункт меню\n");

system("pause");

}

}

}

void load(char\* filename, Circle& circle)

{

circle.isDefined = false;

FILE\* file;

if (fopen\_s(&file, filename, "rb"))

{

cout << "Ошибка открытия файла: " << filename << endl;

system("pause");

return;

}

fread\_s(&circle, sizeof(circle), sizeof(circle), 1, file);

if (circle.isDefined)

{

printf("Данные загружены\n");

if (fclose(file))

{

cout << "Ошибка закрытия файла: " << filename << endl;

}

system("pause");

return;

}

printf("Файл не содержит структуру окружность\n");

if (fclose(file))

{

cout << "Ошибка закрытия файла: " << filename << endl;

}

system("pause");

}

void load(char\* filename, Cube& cube)

{

cube.isDefined = false;

FILE\* file;

if (fopen\_s(&file, filename, "rb"))

{

cout << "Ошибка открытия файла: " << filename << endl;

system("pause");

return;

}

fread\_s(&cube, sizeof(cube), sizeof(cube), 1, file);

if (cube.isDefined)

{

printf("Данные загружены\n");

if (fclose(file))

{

cout << "Ошибка закрытия файла: " << filename << endl;

}

system("pause");

return;

}

printf("Файл не содержит структуру куб\n");

if (fclose(file))

{

cout << "Ошибка закрытия файла: " << filename << endl;

}

system("pause");

}

void load()

{

char filename[100];

while (true)

{

system("cls");

printf("1. Загрузить окружность\n");

printf("2. Загрузить куб\n");

printf("0. В главное меню\n\n");

char menu[80];

cin >> menu;

switch (menu[0])

{

case '1':

printf("Выбранная фигура: окружность\n");

printf("Введите имя файла для загрузки (без .bin): ");

cin >> filename;

strcat\_s(filename, 100, ".bin");

load(filename, circle);

break;

case '2':

printf("Выбранная фигура: куб\n");

printf("Введите имя файла для загрузки (без .bin): ");

cin >> filename;

strcat\_s(filename, 100, ".bin");

load(filename, cube);

break;

case '0': return;

default:

printf("Неправильный пункт меню\n");

system("pause");

}

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

while (true)

{

system("cls");

printf("Главное меню\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("1. Ввести значения фигур\n");

printf("2. Вывод информации о фигуре\n");

printf("3. Сохранение структуры в файл\n");

printf("4. Чтение структуры из файла\n");

printf("0. Выход\n");

char menu[80];

cin >> menu;

switch (menu[0])

{

case '1': setFigure(); break;

case '2': showInfo(); break;

case '3': save(); break;

case '4': load(); break;

case '0':

system("pause");

return 0;

default:

printf("Неправильный пункт меню\n");

system("pause");

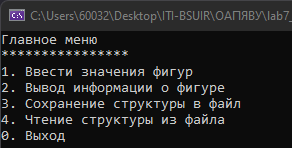
}

}

}

Результат

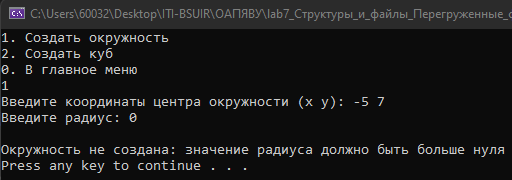
**Главное меню:**



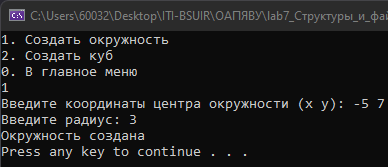
Создание структур

**Создание окружности:**

* Некорректный ввод:

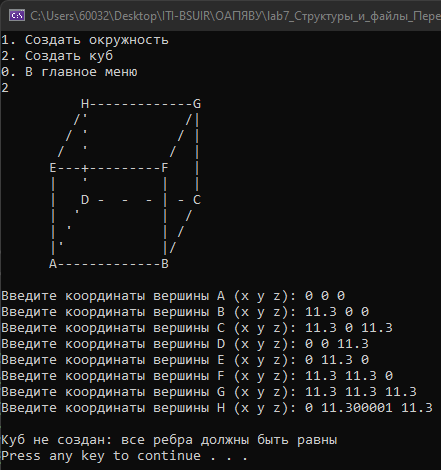


* Корректный ввод:

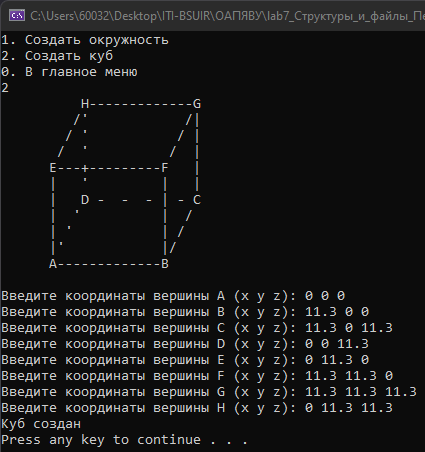


**Создание куба:**

* Некорректный ввод:

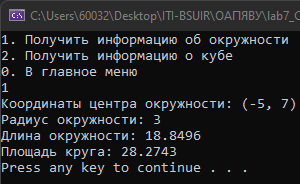
****

* Корректный ввод:

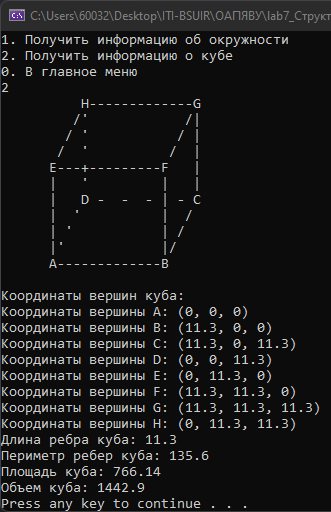


Вывод информации о фигуре на экран

**Вывод информации об окружности:**

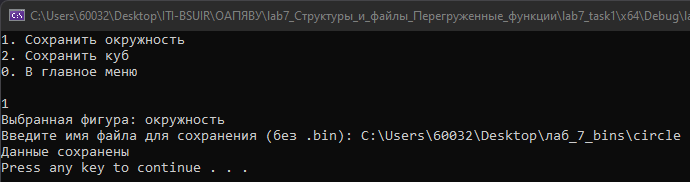
****

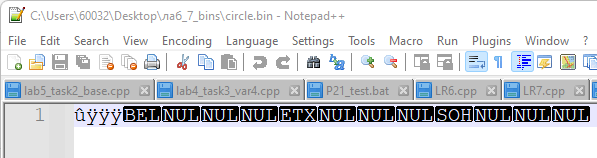
**Вывод информации о кубе:**

****

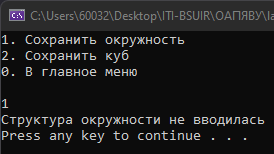
Сохранение структуры в бинарный файл

**Сохранение окружности:**

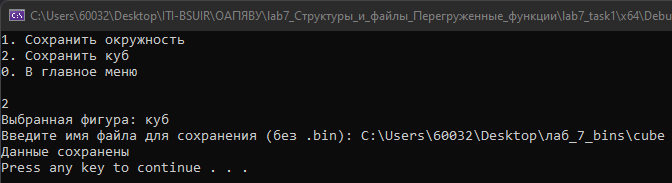
****

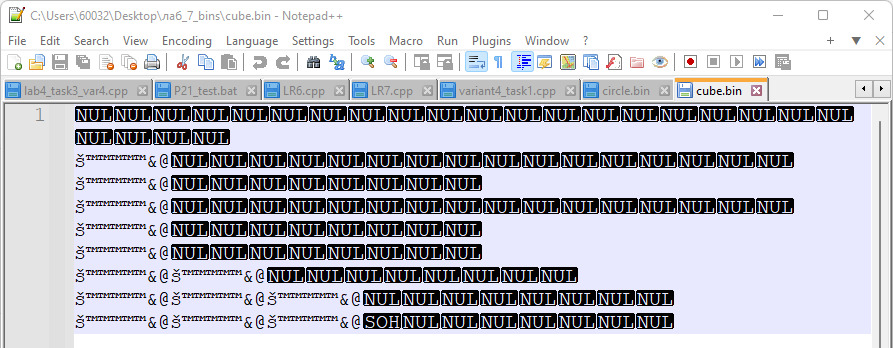
****

Попытка сохранить пустую структуру:

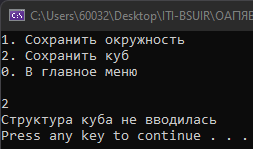


**Сохранение куба:**

****

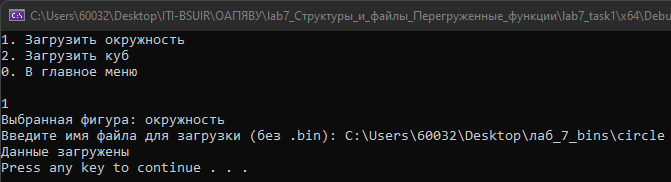
****

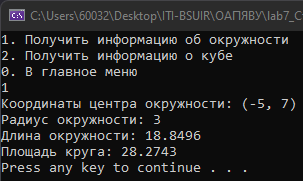
Попытка сохранить пустую структуру:

****

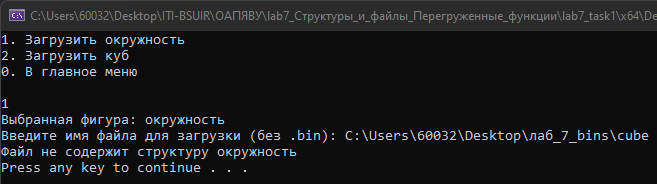
Чтение структуры из файла

**Загрузка окружности:**

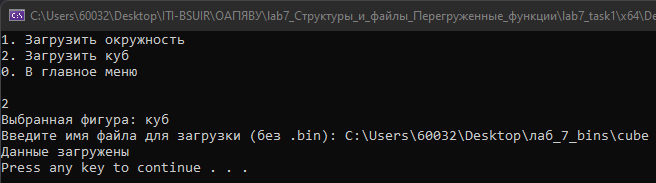
****

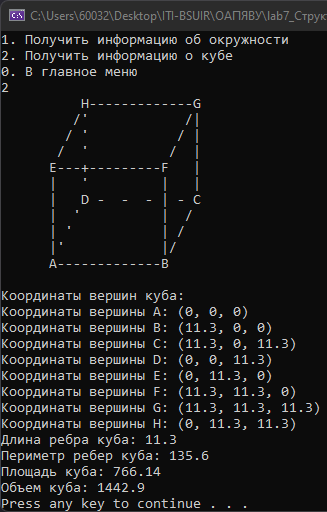
****

Загрузка из файла, не содерщачего структуры окружности:

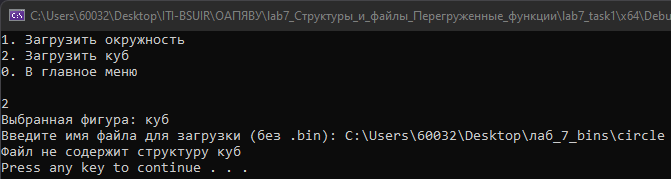


**Загрузка куба:**

****

****

Загрузка из файла, не содерщачего структуры куба:

****

Выводы

Изучены способы создания собственных функций, методы передачи аргументов в функции (их преимущества и недостатки), возможности возврата различных типов значений из функций.