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

**Оглавление**

[ЗАДАНИЕ 1 2](#_Toc161890903)

[Ход выполнения 2](#_Toc161890904)

[Результат 16](#_Toc161890905)

[Создание структур 16](#_Toc161890906)

[Вывод информации о фигуре на экран 18](#_Toc161890907)

[Сохранение структуры в бинарный файл 19](#_Toc161890908)

[Чтение структуры из файла 21](#_Toc161890909)

[ЗАДАНИЕ 2 23](#_Toc161890910)

[Ход выполнения 23](#_Toc161890911)

[Результат 46](#_Toc161890912)

[Вывести данные на экран 46](#_Toc161890913)

[Добавить данные 48](#_Toc161890914)

[Удалить данные 49](#_Toc161890915)

[Редактировать данные 50](#_Toc161890916)

[Записать массив в файл (бинарный) 53](#_Toc161890917)

[Считать массив из файла 54](#_Toc161890918)

[Выполнение задачи 56](#_Toc161890919)

[Проверка исключительных ситуаций 57](#_Toc161890920)

[Выводы 63](#_Toc161890921)

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");

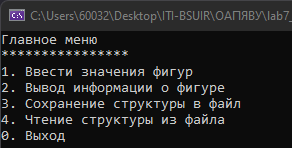
}

}

}

Результат

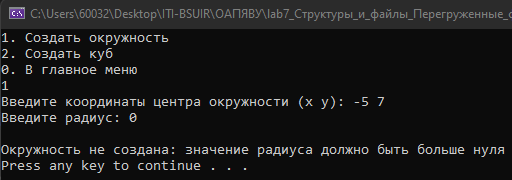
**Главное меню:**



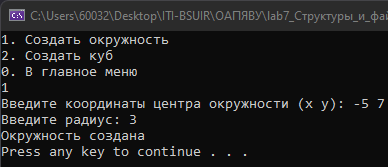
Создание структур

**Создание окружности:**

* Некорректный ввод:

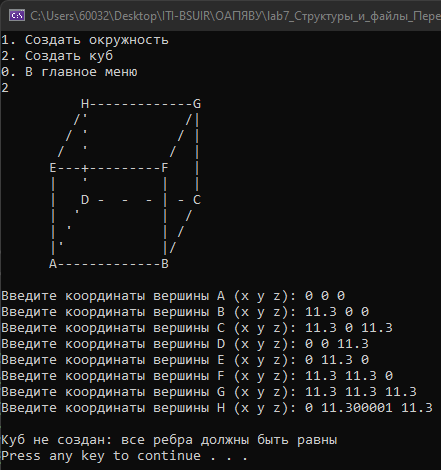


* Корректный ввод:

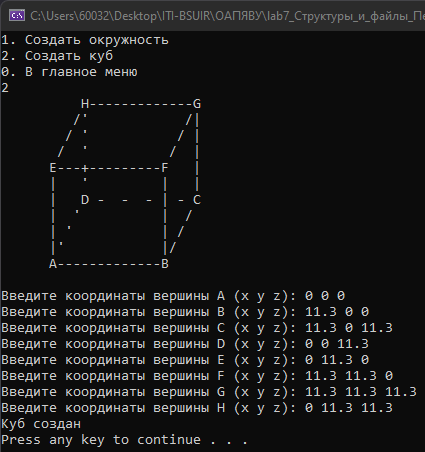


**Создание куба:**

* Некорректный ввод:

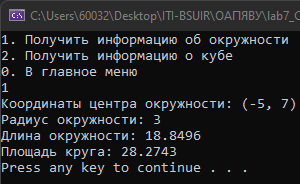
****

* Корректный ввод:

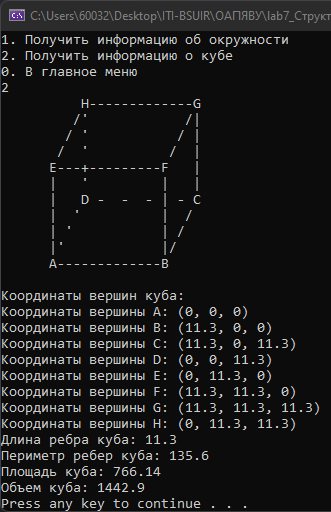


Вывод информации о фигуре на экран

**Вывод информации об окружности:**

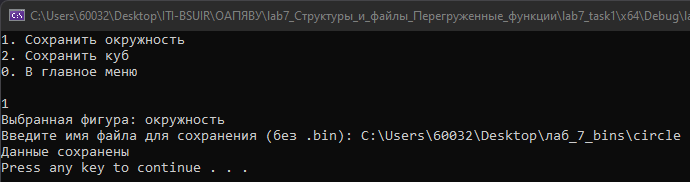
****

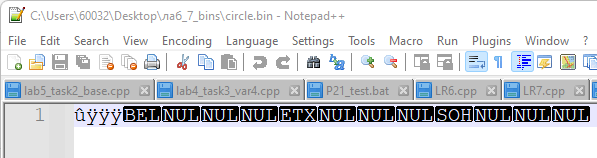
**Вывод информации о кубе:**

****

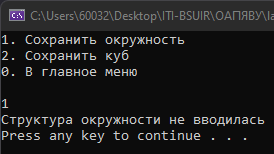
Сохранение структуры в бинарный файл

**Сохранение окружности:**

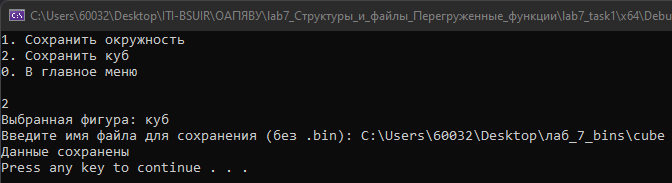
****

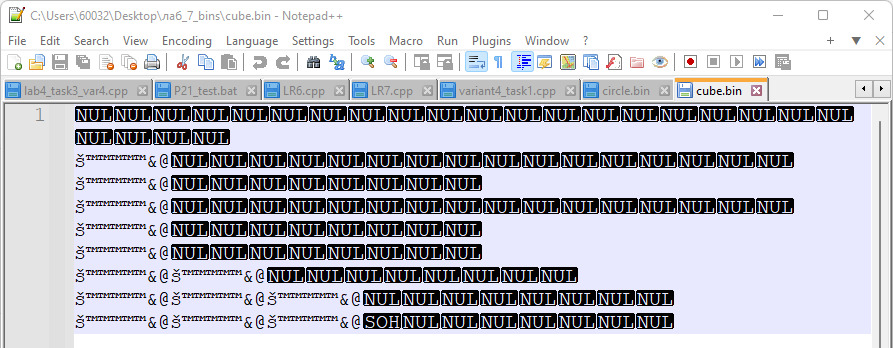
****

Попытка сохранить пустую структуру:

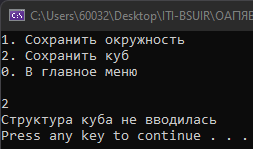


**Сохранение куба:**

****

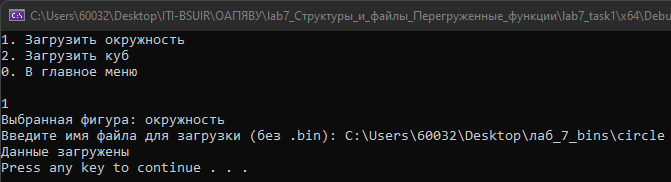
****

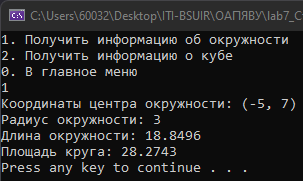
Попытка сохранить пустую структуру:

****

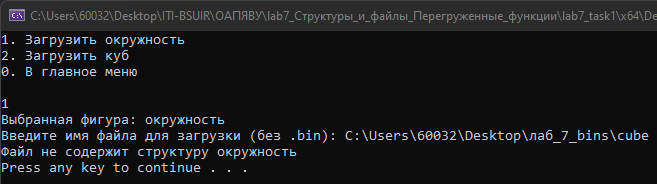
Чтение структуры из файла

**Загрузка окружности:**

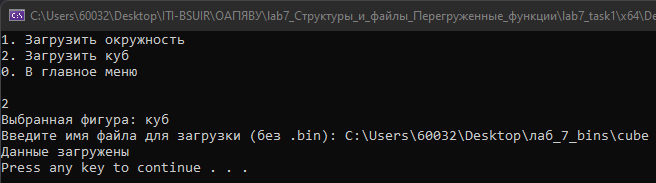
****

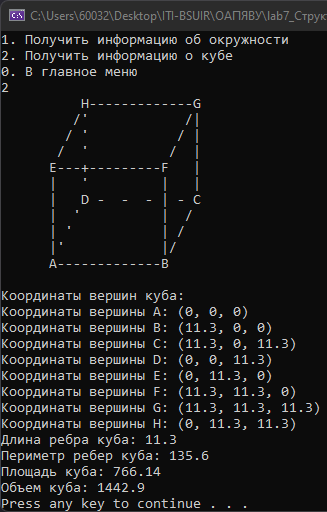
****

Загрузка из файла, не содерщачего структуры окружности:

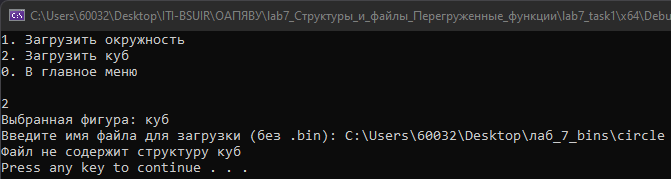


**Загрузка куба:**

****

****

Загрузка из файла, не содерщачего структуры куба:

****

ЗАДАНИЕ 2

**Вариант 4.**

Создать файл, содержащий сведения о сотрудниках фирмы. Запись включает: Ф.И.О; табельный номер; количество проработанных часов за месяц; почасовой тариф для данного сотрудника. Рабочее время свыше 144 часов считается сверхурочным и оплачивается в двойном размере. ВЫВЕСТИ РАЗМЕР ЗАРАБОТНОЙ ПЛАТЫ КАЖДОГО СОТРУДНИКА ФИРМЫ ЗА ВЫЧЕТОМ ПОДОХОДНОГО НАЛОГА, КОТОРЫЙ СОСТАВЛЯЕТ 10% ОТ СУММЫ ЗАРАБОТНОЙ ПЛАТЫ (сортировать по убыванию заработной платы).

Ход выполнения

// c++17

// Lab #7

// task #2

// variant #4

#include <iostream>

#include <Windows.h>

#include <limits>

#include <string>

#include <regex>

#include <filesystem>

#undef max

const int MAX{ 20 }; // Размер массива

const int overWork{ 144 }; // Порог сверхурочного времени

const int incomeTax{ 10 }; // Подоходный налог

const char extension[]{ ".bin" }; // Расширение по умолчанию

int countDefinedEmployees{}; // Счетчик существующих сотрудников

struct Employee

{

// Допустимый размер вводимой строки

const static int cinLineLength{ 30 }; // без '\0'

// Допустимый размер строки

const static int lineLength = cinLineLength + 1; // +'\0'

char lastName[lineLength]{};

char name[lineLength]{};

char patronymic[lineLength]{};

int employeeID{};

int workedHoursPerMonth{};

int hourlyRate{};

double salary{};

};

// Расчет заработной платы

double salary(int workedHours, int hourlyRate)

{

double salary{};

if (workedHours > overWork)

{

salary = overWork \* hourlyRate +

(workedHours - overWork) \* hourlyRate \* 2;

}

else

{

salary = workedHours \* hourlyRate;

}

return (salary / 100) \* (100 - incomeTax);

}

// Очистка failbit

void cinClear()

{

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

// Проверка корректности ввода пункта списка

bool cinOption(int& option)

{

std::cin >> option;

if (!std::cin)

{

printf("Неверный ввод\n");

cinClear();

system("pause");

return false;

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return true;

}

bool noData()

{

if (!countDefinedEmployees)

{

printf("Данные не вводились\n");

system("pause");

return true;

}

return false;

}

// Вывод данных ----------------------------------------

// Вывод всех данных о сотруднике

void employeeData(const Employee& employee)

{

printf("ФИО: %s %s %s\n",

employee.lastName,

employee.name,

employee.patronymic);

printf("Табельный номер: %d\n", employee.employeeID);

printf("Количество проработанных часов за месяц: %d\n",

employee.workedHoursPerMonth);

printf("Почасовой тариф: %d\n", employee.hourlyRate);

printf("Заработная плата: %.2f\n", employee.salary);

}

// Вывод списка сотрудников (ФИО, табельный номер)

void employeesList(Employee\* employees)

{

printf("Список сотрудников\n");

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

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

{

printf("%3d. %s %s %s, табельный номер: %d\n",

i + 1,

employees[i].lastName,

employees[i].name,

employees[i].patronymic,

employees[i].employeeID);

}

}

// Вывод списка сотрудников

// с возможностью просмотра данных о сотруднике

void showEmployeeList(Employee\* employees)

{

int option{};

while (true)

{

system("cls");

printf("\nДля просмотра информации о сотруднике: \n");

printf("Введите номер сотрудника из списка: \n");

printf("0 для выхода\n\n");

employeesList(employees);

if (!cinOption(option)) continue;

if (option < 0 || option > countDefinedEmployees)

{

printf("Неверный ввод\n");

system("pause");

continue;

}

if (!option) return;

system("cls");

employeeData(employees[option - 1]);

system("pause");

}

}

// Показать всех сотрудников с подробными данными

void showAllData(Employee\* employees)

{

system("cls");

printf("Все данные\n");

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

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

{

printf("\n%d. \n", i + 1);

employeeData(employees[i]);

}

system("pause");

}

// Меню вывода данных

void showData(Employee\* employees)

{

int option{};

while (true)

{

system("cls");

printf("Вывод данных\n");

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

printf("1. Вывести список сотрудников\n");

printf("2. Вывести все данные\n");

printf("0. В главное меню\n");

if (!cinOption(option)) continue;

switch (option)

{

case 1:

showEmployeeList(employees);

break;

case 2:

showAllData(employees);

break;

case 0: return;

default:

printf("Неверный ввод\n");

system("pause");

}

}

}

//----------------------------------------------

// Добавление данных ---------------------------

// Предложение ввода фамилии, имени, отчества

bool offerInput(const char\* message, char\* data, bool& isContinue)

{

printf("\n%s (0 - отмена): ", message);

std::cin.getline(data, Employee::lineLength);

if (!std::cin && std::cin.gcount() == Employee::cinLineLength)

{

printf("Превышено допустимое количество вводимых символов: %d\n",

Employee::cinLineLength);

cinClear();

return false;

}

if (data[0] == '0' && strlen(data) == 1) isContinue = false;

return true;

}

// Предложение ввода количества проработанных часов за месяц,

// почасового тарифа

bool offerInput(const char\* message, int& data, bool& isContinue)

{

printf("\n%s\n(отрицательное число - отмена): ", message);

std::cin >> data;

if (!std::cin)

{

printf("Введено не число\n");

cinClear();

return false;

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

if (data < 0) isContinue = false;

return true;

}

// Меню добавления данных

bool addDataMenu()

{

int option{};

while (true)

{

system("cls");

printf("Добавить нового сотрудника\n");

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

printf("1. Ввести данные\n");

printf("0. В главное меню\n");

if (!cinOption(option)) continue;

switch (option)

{

case 1: return true;

case 0: return false;

default:

printf("Неверный ввод\n");

system("pause");

}

}

}

// Добавление данных

void addData(Employee\* employees)

{

if (countDefinedEmployees == MAX)

{

printf("Штат заполнен\n");

printf("Невозможно добавить нового сотрудника\n");

system("pause");

return;

}

bool isContinue{};

int i{};

while (true)

{

i = countDefinedEmployees;

isContinue = true;

if (!addDataMenu()) return;

system("cls");

printf("\nДопустимое количество вводимых символов: %d\n", Employee::cinLineLength);

while (!offerInput("Введите фамилию", employees[i].lastName, isContinue));

if (!isContinue) continue;

while (!offerInput("Введите имя", employees[i].name, isContinue));

if (!isContinue) continue;

while (!offerInput("Введите отчество", employees[i].patronymic, isContinue));

if (!isContinue) continue;

while (!offerInput("Введите количество проработанных часов за месяц",

employees[i].workedHoursPerMonth, isContinue));

if (!isContinue) continue;

while (!offerInput("Введите почасовой тариф",

employees[i].hourlyRate, isContinue));

if (!isContinue) continue;

employees[i].salary =

salary(employees[i].workedHoursPerMonth, employees[i].hourlyRate);

if (employees[0].employeeID)

{

employees[i].employeeID = employees[i - 1].employeeID + 1;

}

else

{

employees[i].employeeID = 1;

}

countDefinedEmployees++;

printf("\nСотрудник добавлен\n");

system("pause");

}

}

// Заполнение массива начальными данными

void fillEmloyeesArray(Employee\* employees, int size)

{

if (size > MAX)

{

printf("size > MAX\n");

system("pause");

exit(2);

}

if (size < 0)

{

printf("size < 0\n");

system("pause");

exit(2);

}

srand(static\_cast<unsigned int>(time(0)));

int minHour{ 130 };

int maxHour{ 160 };

int minRate{ 10 };

int maxRate{ 20 };

std::string names[20][3] =

{

{"Борисова", "Александра", "Артёмовна"},

{"Королева", "Анна", "Михайловна"},

{"Яковлев", "Лев", "Михайлович"},

{"Киселев", "Михаил", "Лукич" },

{"Калашникова", "Алёна", "Богдановна"},

{"Платонова", "Анна", "Филипповна"},

{"Кузнецова", "Анна", "Фёдоровна"},

{"Зеленина", "Мария", "Фёдоровна"},

{"Соболева", "Екатерина", "Арсентьевна"},

{"Беляева", "Анна", "Алексеевна"},

{"Ермаков", "Артём", "Борисович"},

{"Корнилов", "Владимир", "Иванович"},

{"Бобров", "Матвей", "Мирославович"},

{"Усова", "Олеся", "Андреевна"},

{"Раков", "Артём", "Даниилович"},

{"Васильев", "Даниил", "Вячеславович"},

{"Крылов", "Егор", "Георгиевич"},

{"Журавлев", "Андрей", "Денисович"},

{"Федотова", "Оливия", "Дмитриевна"},

{"Дубровин", "Александр", "Ярославович"}

};

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

{

strcpy\_s(employees[i].lastName, names[i][0].c\_str());

strcpy\_s(employees[i].name, names[i][1].c\_str());

strcpy\_s(employees[i].patronymic, names[i][2].c\_str());

employees[i].employeeID = i + 1;

employees[i].workedHoursPerMonth =

minHour + rand() % (maxHour - minHour + 1);

employees[i].hourlyRate =

minRate + rand() % (maxRate - minRate + 1);

employees[i].salary =

salary(employees[i].workedHoursPerMonth, employees[i].hourlyRate);

countDefinedEmployees++;

}

}

// ---------------------------------------

// Удаленее данных -----------------------

Employee\* deleteData(Employee\* employees, int index)

{

Employee\* tempEmployees{ nullptr };

try

{

tempEmployees = new Employee[MAX];

}

catch (const std::bad\_alloc& e)

{

std::cout << "Выделение памяти не удалось: " << e.what() << std::endl;

printf("Данные не удалены\n");

system("pause");

exit(1);

}

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

{

tempEmployees[i] = employees[i];

}

for (int i = 0; i < countDefinedEmployees - index - 1; index++)

{

tempEmployees[index] = employees[index + 1];

}

countDefinedEmployees--;

return tempEmployees;

}

Employee\* deleteData(Employee\* employees)

{

int option{};

int index{};

while (true)

{

if (!countDefinedEmployees)

{

printf("\nДанных больше не существует\n");

system("pause");

return employees;

}

system("cls");

printf("Удаление сотрудника\n");

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

printf("\nДля удаления сотрудника: \n");

printf("Введите номер сотрудника из списка\n");

printf("0 для выхода\n\n");

employeesList(employees);

if (!cinOption(option)) continue;

if (option < 0 || option > countDefinedEmployees)

{

printf("Неверный ввод\n");

system("pause");

continue;

}

index = option - 1;

if (!option) return employees;

printf("Вы собираетесь удалить сотрудника:\n%s %s %s\n",

employees[index].lastName,

employees[index].name,

employees[index].patronymic);

printf("Для подтверждения введите: 1\n");

printf("Отмена: любой другой символ\n");

std::cin >> option;

if (!std::cin)

{

cinClear();

continue;

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

if (option == 1)

{

employees = deleteData(employees, index);

printf("Сотрудник удален\n");

system("pause");

}

}

}

// --------------------------------------------------

// Редактирование данных ----------------------------

void editData(Employee& employee)

{

Employee baseEmployee = employee;

int option{};

while (true)

{

system("cls");

employeeData(employee);

printf("\n1. Изменить фамилию\n");

printf("2. Изменить имя\n");

printf("3. Изменить отчество\n");

printf("4. Изменить количество проработанных часов за месяц\n");

printf("5. Изменить почасовой тариф\n");

printf("0. Выход\n");

if (!cinOption(option)) continue;

try

{

; switch (option)

{

case 1:

printf("Введите новую фамилию (не более %d символов): ",

Employee::cinLineLength);

std::cin.getline(employee.lastName, Employee::lineLength);

if (!std::cin && std::cin.gcount() == Employee::cinLineLength)

{

strcpy\_s(employee.lastName, baseEmployee.lastName);

throw 1;

}

printf("Фамилия изменена\n");

system("pause");

break;

case 2:

printf("Введите новое имя (не более %d символов): ",

Employee::cinLineLength);

std::cin.getline(employee.name, Employee::lineLength);

if (!std::cin && std::cin.gcount() == Employee::cinLineLength)

{

strcpy\_s(employee.name, baseEmployee.name);

throw 1;

}

printf("Имя изменено\n");

system("pause");

break;

case 3:

printf("Введите новое отчество (не более %d символов): ",

Employee::cinLineLength);

std::cin.getline(employee.patronymic, Employee::lineLength);

if (!std::cin && std::cin.gcount() == Employee::cinLineLength)

{

strcpy\_s(employee.patronymic, baseEmployee.patronymic);

throw 1;

}

printf("Отчество изменено\n");

system("pause");

break;

case 4:

printf("Введите количество проработанных часов: ");

std::cin >> employee.workedHoursPerMonth;

if (!std::cin || employee.workedHoursPerMonth < 0)

{

employee.workedHoursPerMonth = baseEmployee.workedHoursPerMonth;

throw 2;

}

printf("Количество проработанных часов изменено\n");

employee.salary =

salary(employee.workedHoursPerMonth, employee.hourlyRate);

system("pause");

break;

case 5:

printf("Введите новый почасовой тариф: ");

std::cin >> employee.hourlyRate;

if (!std::cin || employee.hourlyRate < 0)

{

employee.hourlyRate = baseEmployee.hourlyRate;

throw 2;

}

printf("Почасовой тариф изменен\n");

employee.salary =

salary(employee.workedHoursPerMonth, employee.hourlyRate);

system("pause");

break;

case 0: return;

default:

printf("Неверный ввод\n");

system("pause");

}

}

catch (int i)

{

switch (i)

{

case 1:

printf("Превышено допустимое количество вводимых символов: %d\n", Employee::cinLineLength);

cinClear();

system("pause");

break;

case 2:

printf("Введено не число, либо отрицательное число\n");

cinClear();

system("pause");

break;

}

}

baseEmployee = employee;

}

}

void editData(Employee\* employees)

{

int option{};

int index{};

while (true)

{

system("cls");

printf("Редактирование данных сотрудника\n");

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

printf("\nДля редактирования данных сотрудника: \n");

printf("Введите номер сотрудника из списка: \n");

printf("0 для выхода\n\n");

employeesList(employees);

if (!cinOption(option)) continue;

if (option < 0 || option > countDefinedEmployees)

{

printf("Неверный ввод\n");

system("pause");

continue;

}

if (!option) return;

index = option - 1;

editData(employees[index]);

}

}

// ----------------------------------------

// Формирование пути имени файла

bool pathParcing(std::string& path, const bool createFolder)

{

std::cmatch cm{};

char directory[MAX\_PATH]{};

std::string currentPath{ std::filesystem::current\_path().string() };

for (auto& symbol : currentPath)

{

if (symbol == '\\') symbol = '/';

}

std::cin.getline(directory, MAX\_PATH);

if (!std::cin && std::cin.gcount() == MAX\_PATH - 1) // - '\0'

{

printf("Превышена максимальная длина пути: %d\n", MAX\_PATH);

cinClear();

system("pause");

return false;

}

for (int i = 0; i < strlen(directory); i++)

{

if (directory[i] == '\\') directory[i] = '/';

}

do

{

// Игнорирование двойных кавычек

// Игнорирование символа '/', если введен последним

if (std::regex\_search(directory, cm, std::regex("[^\"]\*[^\"/]")))

{

if (cm.str()[0] == '/')

{

path = currentPath[0];

path += (':' + cm.str());

break;

}

if (cm.str()[1] == ':')

{

path = cm.str();

break;

}

path = currentPath + '/' + cm.str();

break;

}

if (cm.empty())

{

path = currentPath;

break;

}

} while (false);

if (path.size() > MAX\_PATH - 1) // - '\0'

{

printf("Превышена максимальная длина пути: %d\n", MAX\_PATH);

system("pause");

return false;

}

// Выполняется, если нужно создать новую папку

if (!createFolder) return true;

if (!std::filesystem::exists(path))

{

printf("Такой директории не существует.\n");

printf("Создать директорию?\n");

printf("Да: введите 1\n");

printf("Нет: любой другой символ\n");

int option{};

std::cin >> option;

if (!std::cin)

{

cinClear();

return false;

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

if (option != 1) return false;

try

{

if (!std::filesystem::create\_directories(path))

{

printf("Не удалось создать директорию\n");

system("pause");

return false;

}

}

catch (std::filesystem::filesystem\_error const& e)

{

std::cout << "Не удалось создать директорию:\n" << e.what() << std::endl;

system("pause");

return false;

}

}

return true;

}

// Проверка корректности ввода имени файла

// Допустимо вводить с расширением ('.' и следующие за ней символы игнорируются)

bool filenameInput(char\* filename, int maxFileNameLength)

{

char ch{};

int i{};

for (; i < maxFileNameLength - strlen(extension); i++)

{

std::cin.get(ch);

if (ch == '\n')

{

filename[i] = '\0';

strcat\_s(filename, maxFileNameLength, extension);

return true;

}

if (ch == '.')

{

filename[i] = '\0';

strcat\_s(filename, maxFileNameLength, extension);

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return true;

}

if (ch == '\\'

|| ch == '/'

|| ch == ':'

|| ch == '\*'

|| ch == '?'

|| ch == '\"'

|| ch == '<'

|| ch == '>'

|| ch == '|')

{

printf("Имя файла не может содержать символы: \\ / : \* ? \" < > |\n");

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return false;

}

filename[i] = ch;

}

printf("Превышена максимальная длина имени файла\n");

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return false;

}

// Запись массива в файл (бинарный) -----------------------------------

void save(Employee\* employees)

{

const bool createFolder{ true };

int maxFileNameLength{};

std::string path{};

printf("Введите путь директории для сохранения:\n");

printf("(Enter - сохранить в текущей директории): ");

std::cout << std::filesystem::current\_path().string() << std::endl;

if (!pathParcing(path, createFolder)) return;

maxFileNameLength = MAX\_PATH - static\_cast<int>(path.size());

char\* cinFilename = new char[maxFileNameLength];

std::cout << "Введите имя файла для сохранения "

<< "(не более " << maxFileNameLength - strlen(extension) << " символов): "

<< std::endl;

if (!filenameInput(cinFilename, maxFileNameLength))

{

printf("Данные не сохранены\n");

system("pause");

delete[] cinFilename;

return;

}

path += '/';

path += cinFilename;

delete[] cinFilename;

cinFilename = nullptr;

const char\* filename{ path.c\_str() };

char\* b{ nullptr };

FILE\* file;

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

{

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

{

printf("Ошибка открытия файла: %s\n", filename);

system("pause");

return;

}

}

else

{

printf("Файл с таким именем уже существует: %s\n", filename);

printf("Перезаписать?:\n");

printf("Да: введите 1\n");

printf("Нет: любой другой символ\n");

int option{};

std::cin >> option;

if (!std::cin)

{

cinClear();

if (fclose(file))

{

printf("Ошибка закрытия файла: %s\n", filename);

system("pause");

}

return;

}

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

if (option != 1)

{

if (fclose(file))

{

printf("Ошибка закрытия файла: %s\n", filename);

system("pause");

}

return;

}

}

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

{

b = (char\*)&employees[i];

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

{

if (fputc(\*(b++), file) == EOF)

{

printf("Ошибка записи в файл\n");

if (fclose(file))

{

printf("Ошибка закрытия файла: %s\n", filename);

}

system("pause");

return;

}

}

}

for (auto& symbol : path)

{

if (symbol == '/') symbol = '\\';

}

std::cout << "Данные сохранены в: "

<< std::filesystem::path(path).string() << std::endl;

if (fclose(file))

{

printf("Ошибка закрытия файла: %s\n", filename);

}

system("pause");

}

// ----------------------------------------------------------

// Считывание массива из файла ------------------------------

bool load(Employee\* employees)

{

const bool createFolder{ false };

int maxFileNameLength{};

std::string path{};

printf("Введите директорию для загрузки:\n");

printf("(Enter - загрузить из текущей директории): ");

std::cout << std::filesystem::current\_path().string() << std::endl;

if (!pathParcing(path, createFolder)) return false;

if (!std::filesystem::exists(path))

{

printf("Такой директории не существует.\n");

system("pause");

return false;

}

maxFileNameLength = MAX\_PATH - static\_cast<int>(path.size());

char\* cinFilename = new char[maxFileNameLength];

std::cout << "Введите имя файла для загрузки "

<< "(не более " << maxFileNameLength - strlen(extension) << " символов): "

<< std::endl;

if (!filenameInput(cinFilename, maxFileNameLength))

{

printf("Данные не загружены\n");

system("pause");

delete[] cinFilename;

return false;

}

path += '/';

path += cinFilename;

delete[] cinFilename;

cinFilename = nullptr;

const char\* filename{ path.c\_str() };

FILE\* file;

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

{

printf("Ошибка открытия файла: %s\n", filename);

system("pause");

return false;

}

if (

fread\_s(

employees,

sizeof(Employee) \* MAX,

sizeof(Employee),

MAX,

file)

< MAX)

{

printf("Данные считаны неверно\n");

if (fclose(file))

{

printf("Ошибка закрытия файла: %s\n", filename);

}

system("pause");

return false;

}

printf("Данные загружены\n");

if (fclose(file))

{

printf("Ошибка закрытия файла: %s\n", filename);

}

system("pause");

return true;

}

// ----------------------------------------------------------

// Выполнение задачи (вариант: 4) ---------------------------

/\* Рабочее время свыше 144 часов

считается сверхурочным и оплачивается в двойном размере.

ВЫВЕСТИ РАЗМЕР ЗАРАБОТНОЙ ПЛАТЫ

КАЖДОГО СОТРУДНИКА ФИРМЫ ЗА ВЫЧЕТОМ

ПОДОХОДНОГО НАЛОГА, КОТОРЫЙ СОСТАВЛЯЕТ

10 % ОТ СУММЫ ЗАРАБОТНОЙ ПЛАТЫ

(сортировать по убыванию заработной платы). \*/

// Функция сортировки

void selectionSort(Employee\* salaries)

{

int maxSalaryIndex{};

Employee tempEmployee{};

for (int i = 0; i < countDefinedEmployees - 1; i++)

{

maxSalaryIndex = i;

for (int j = i + 1; j < countDefinedEmployees; j++)

{

if (salaries[j].salary > salaries[maxSalaryIndex].salary)

{

maxSalaryIndex = j;

}

}

if (maxSalaryIndex != i)

{

tempEmployee = salaries[i];

salaries[i] = salaries[maxSalaryIndex];

salaries[maxSalaryIndex] = tempEmployee;

}

}

}

void sortSalaries(Employee\* employees)

{

Employee\* salaries{ nullptr };

try

{

salaries = new Employee[countDefinedEmployees];

}

catch (const std::bad\_alloc& e)

{

std::cout << "Выделение памяти не удалось: " << e.what() << std::endl;

printf("Динамический массив не создан\n");

system("pause");

exit(1);

}

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

{

salaries[i] = employees[i];

}

selectionSort(salaries);

system("cls");

printf("Сортированный список заработных плат (по убыванию)\n");

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

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

{

printf("%s %s %s\n",

salaries[i].lastName,

salaries[i].name,

salaries[i].patronymic);

printf("Заработная плата: %.2f\n\n", salaries[i].salary);

}

delete[] salaries;

system("pause");

}

// --------------------------------------------------------------

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Employee\* employees{ nullptr };

Employee\* tempEmployees{ nullptr };

try

{

employees = new Employee[MAX];

}

catch (const std::bad\_alloc& e)

{

std::cout << "Выделение памяти не удалось: " << e.what() << std::endl;

printf("Динамический массив не создан\n");

system("pause");

return 1;

}

fillEmloyeesArray(employees, 0);

while (true)

{

system("cls");

printf("Главное меню\n");

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

printf("1. Вывести данные на экран\n");

printf("2. Добавить данные\n");

printf("3. Удалить данные\n");

printf("4. Редактировать данные\n");

printf("5. Записать массив в файл (бинарный)\n");

printf("6. Считать массив из файла\n");

printf("7. Выполнение задачи\n");

printf("0. Выход\n");

int option{};

if (!cinOption(option)) continue;

switch (option)

{

case 1:

if (noData()) break;

showData(employees); break;

case 2: addData(employees); break;

case 3:

if (noData()) break;

tempEmployees = deleteData(employees);

if (tempEmployees == employees) break;

delete[] employees;

employees = tempEmployees;

break;

case 4:

if (noData()) break;

editData(employees);

break;

case 5:

if (noData()) break;

save(employees);

break;

case 6:

if (load(employees))

{

for (int i = MAX - 1; i >= 0; i--)

{

if (employees[i].employeeID)

{

countDefinedEmployees = i + 1;

break;

}

}

}

break;

case 7:

if (noData()) break;

sortSalaries(employees);

break;

case 0:

system("pause");

return 0;

default:

printf("Неверный ввод\n");

system("pause");

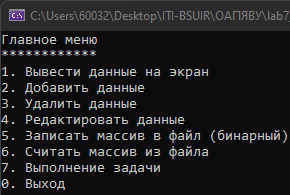
}

}

}

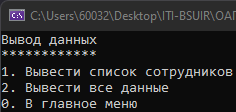
Результат

Пользовательское меню:

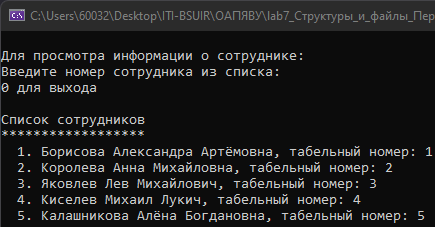


Вывести данные на экран

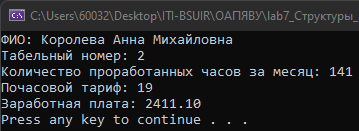
Меню вывода данных (начальные данные заполняются функцией fillEmloyeesArray):



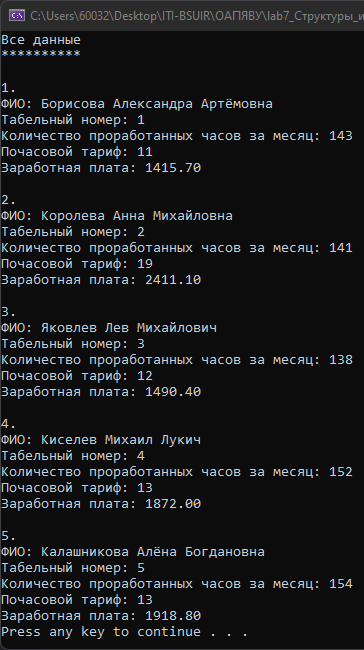
Вывод списка сотрудников:



Вывод информации о сотруднике:

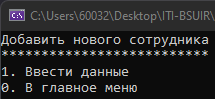


Вывод всех данных:

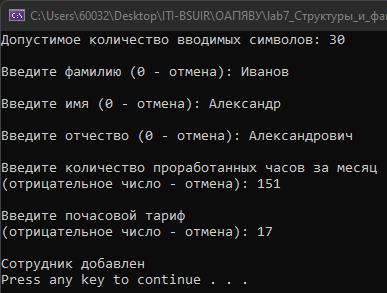


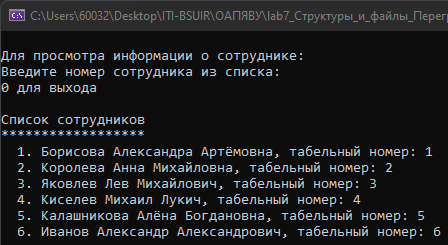
Добавить данные

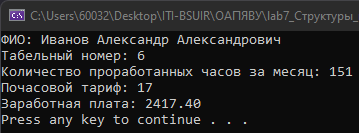
Меню добавления данных:



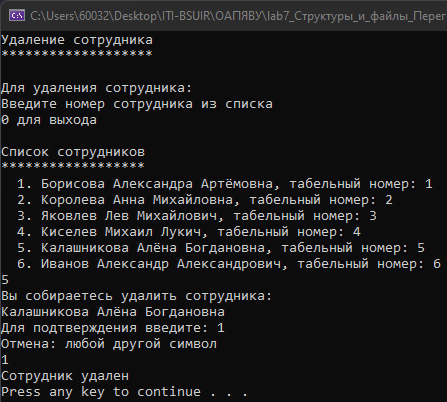
Добавление данных:

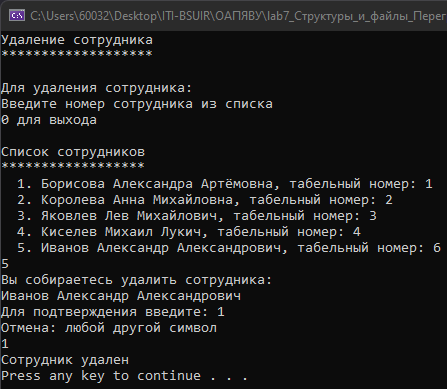


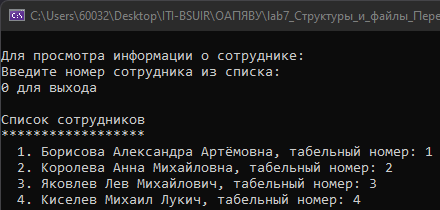




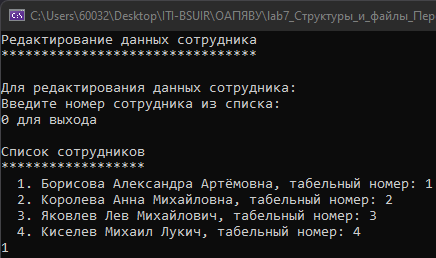
Удалить данные

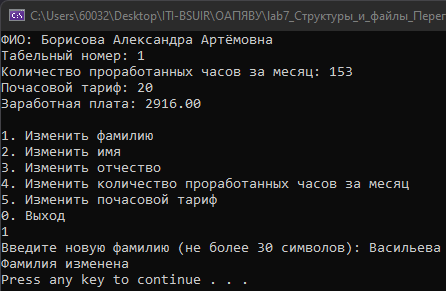


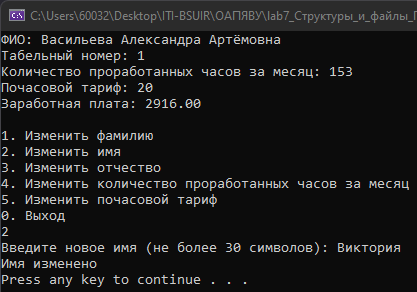


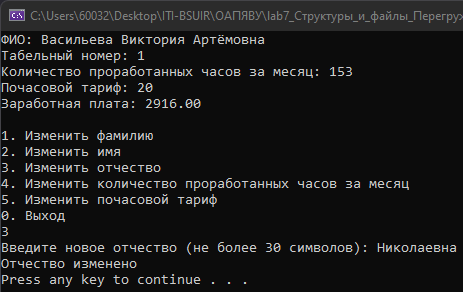


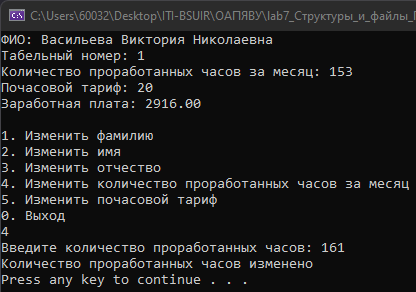
Редактировать данные

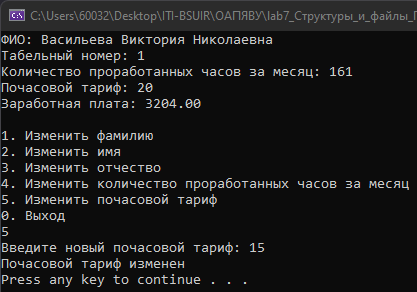


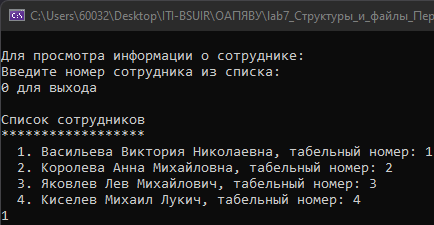


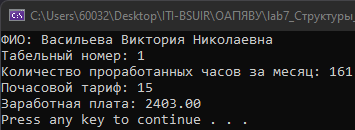




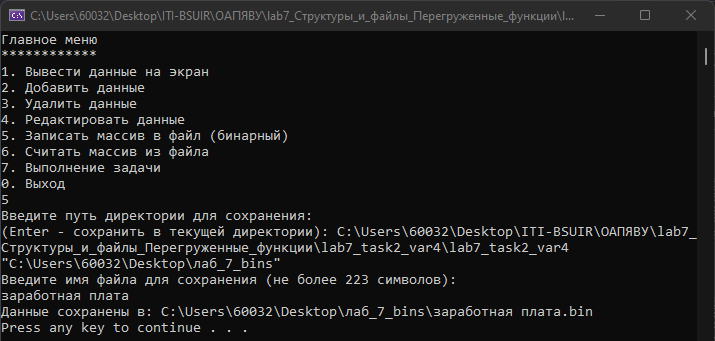


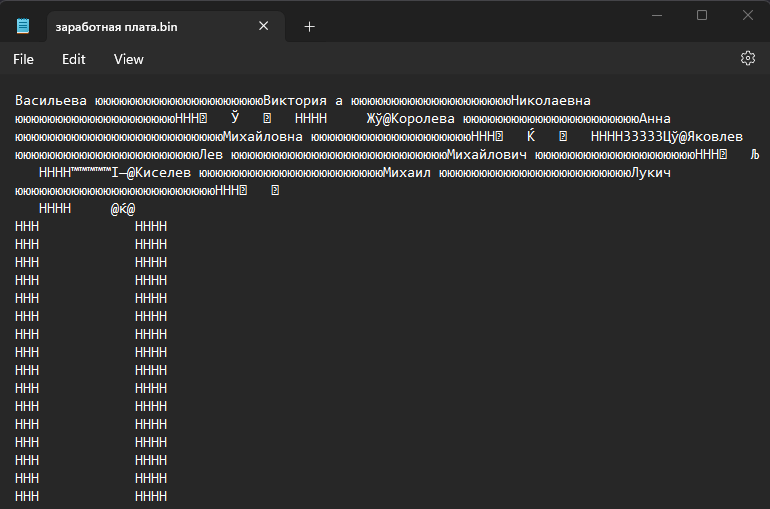




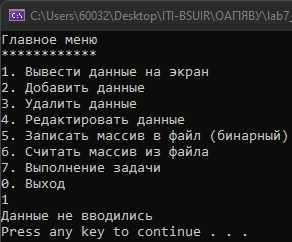


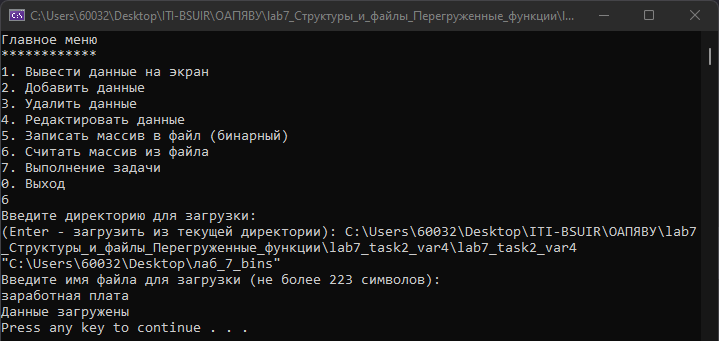
Записать массив в файл (бинарный)

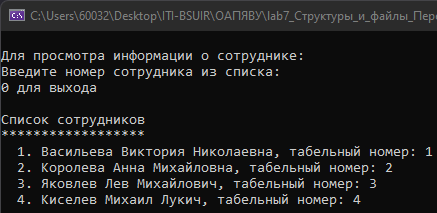


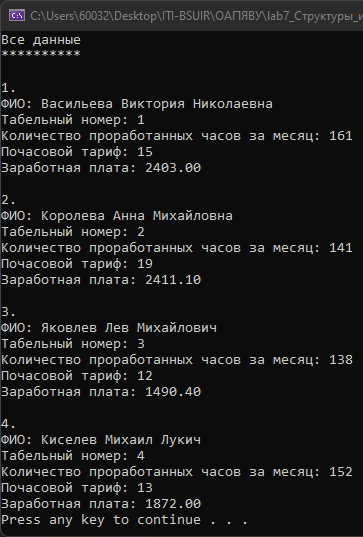


Считать массив из файла





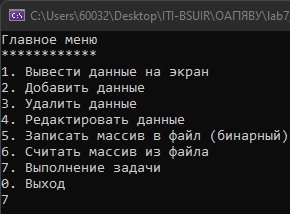


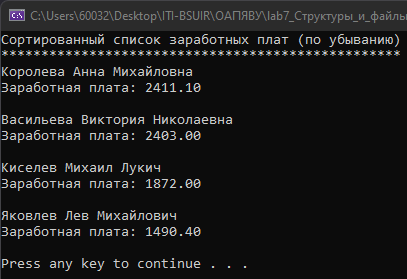


Выполнение задачи

**Вариант 4.**

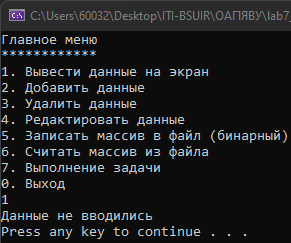
Создать файл, содержащий сведения о сотрудниках фирмы. Запись включает: Ф.И.О; табельный номер; количество проработанных часов за месяц; почасовой тариф для данного сотрудника. Рабочее время свыше 144 часов считается сверхурочным и оплачивается в двойном размере. ВЫВЕСТИ РАЗМЕР ЗАРАБОТНОЙ ПЛАТЫ КАЖДОГО СОТРУДНИКА ФИРМЫ ЗА ВЫЧЕТОМ ПОДОХОДНОГО НАЛОГА, КОТОРЫЙ СОСТАВЛЯЕТ 10% ОТ СУММЫ ЗАРАБОТНОЙ ПЛАТЫ (сортировать по убыванию заработной платы).

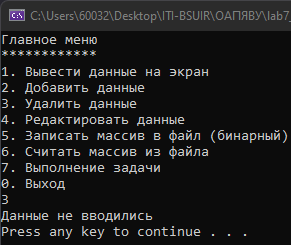


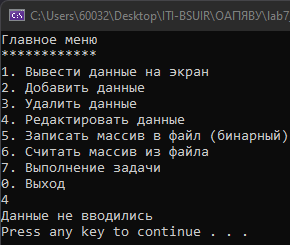


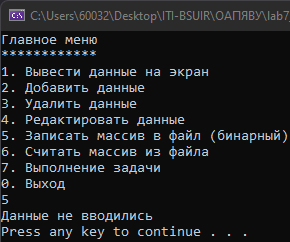
Проверка исключительных ситуаций

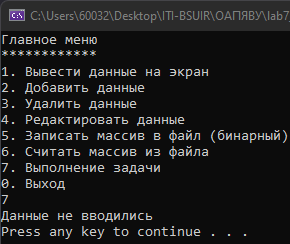
Проверка наличия пустого массива:



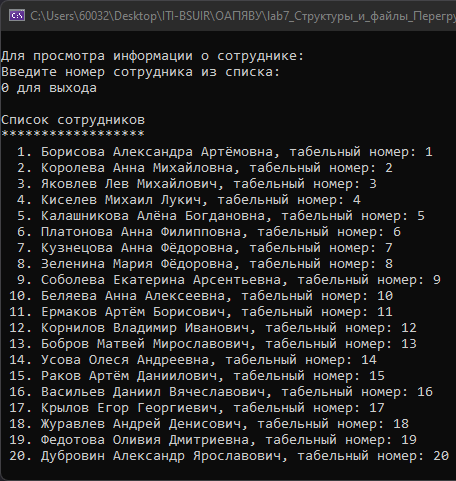


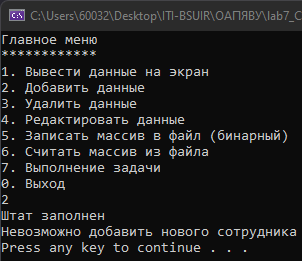




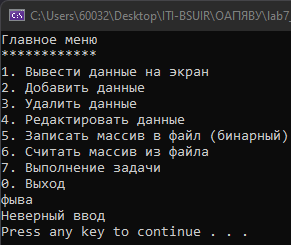


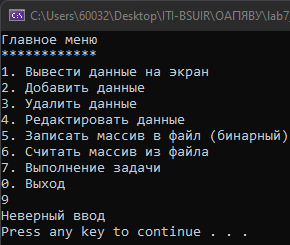
Попытка переполнения массива:



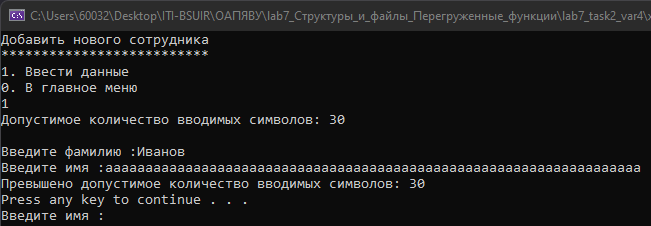


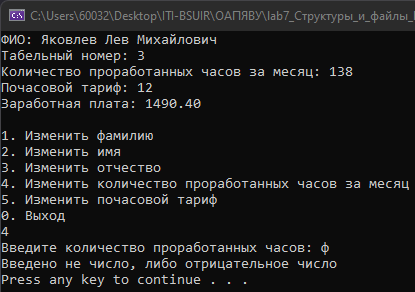
Некорректный ввод пункта меню:

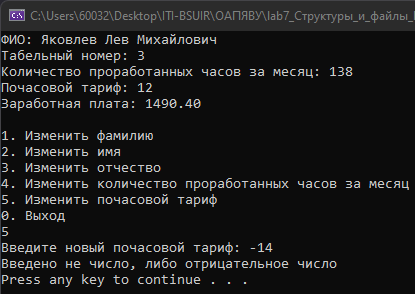




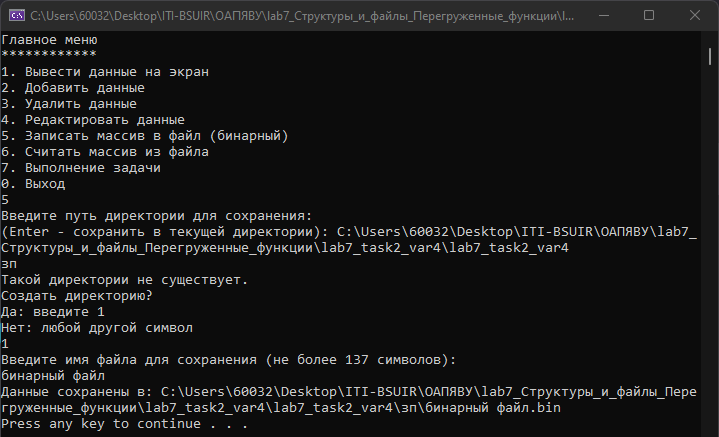
Некорректный ввод при добавлении/редактировании данных:



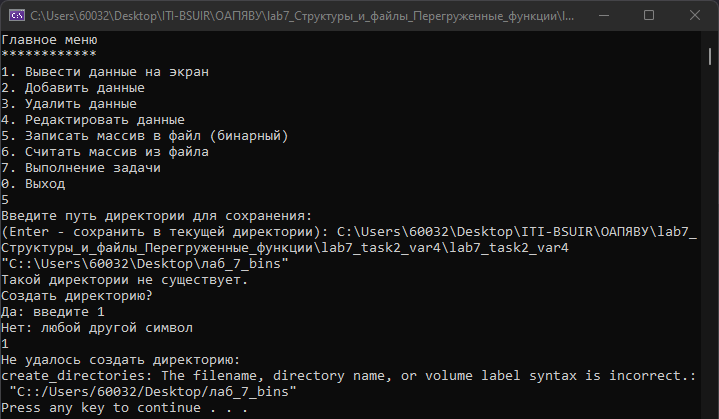




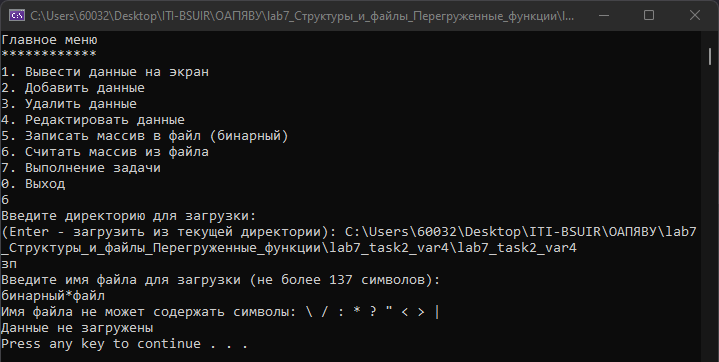
Отсутствие директории для сохранения:



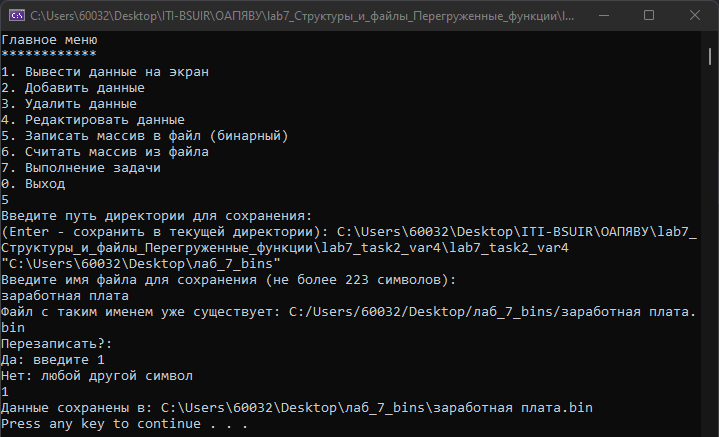
Некореектный ввод пути:



Некорректный ввод имени файла:



Попытка перезаписи файла:



Выводы

Получены практические навыки работы со структурами и бинарными файлами:

* создание структур;
* создание массива структур;
* сохранение структур, массива структур в бинарный файл;
* считывание этих структур из бинарного файл.

Изучены способы обработки исключительных ситуаций.

Также применена на практике перегрузка функций.