МИНОБРНАУКИ РОССИИ

Федеральные государственное бюджетное образовательное учреждение высшего образования

«Чувашский государственный университет имени И. Н. Ульянова»

Факультет информатики и вычислительной техники

Кафедра компьютерных технологий

РАСЧЁТНО-ГРАФИЧЕСАКЯ РАБОТА

По дисциплине «Программирование»

Выполнил: студент гр. КТ-31-24

Киселев М.А

Проверила: Давыдова О.В

№\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_

Чебоксары

2025

**Структуры данных:**

// находится в Order.h

// структура заказа  
struct Order {

int id;

int clientId;

int driverId;

Address shippingAddress;

Address recipientAddress;

int routeLength;

std::time\_t date;

};

// находится в Address.h

// структура адреса

struct Address {

std::string city;

std::string street;

std::string house;

std::string entrance;

std::string apartment;

};

// находится в Client.h

// структура клиента  
struct Client {

int id;

std::string clientName;

std::string firstName;

std::string lastName;

int dateOfBirth[3];

std::vector<int> orderHistory;

};

// находится в Driver.h

// структура водителя  
struct Driver {

int id;

std::string driverName;

std::string firstName;

std::string lastName;

int dateOfBirth[3];

int drivingExperience;

Car car;

std::vector<int> orderHistory;

};

// находится в Car.h

// структура автомобиля

struct Car {

std::string brand;

std::string model;

std::string classC;

int year;

};

**Главный файл:**

// находится в main.cpp  
#include <iostream>

#include <string>

#include "../Header Files/InitData/OrderSeed.h"

#include "../Header Files/InitData/ClientSeed.h"

#include "../Header Files/InitData/DriverSeed.h"

#include "../Header Files/UserInterface/UserInterface.h"

int main() {

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

setlocale(LC\_ALL, "rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::string mode;

std::cout << "Введите \"Enter\" чтобы продолжить..."; std::getline(std::cin, mode);

if (mode.empty()) {

menu();

}

else if (mode == "seedDataBase") {

deleteDataInClientDataBase();

initializationClientDataBase();

deleteDataInDriverDataBase();

initializationDriverDataBase();

deleteDataInOrderDataBase();

initializationOrderDataBase();

}

else {

std::cout << "Error!" << std::endl;

}

return 0;

}

**Основной интерфейс:**

// находится в UserInterface.h  
#ifndef USERINTERFACE\_H

#define USERINTERFACE\_H

void orderMenu();

void clientMenu();

void driverMenu();

void exitMenu();

void menu();

#endif

// находится в UserInterface.cpp   
#include <cstdlib>

#include <functional>

#include <iostream>

#include <map>

#include "../../Header Files/Services/ClientService.h"

#include "../../Header Files/Services/DriverService.h"

#include "../../Header Files/UserInterface/UserInterface.h"

#include "../../Header Files/UserInterface/Interfaces/OrderInterface.h"

#include "../../Header Files/UserInterface/Interfaces/ClientInterface.h"

#include "../../Header Files/UserInterface/Interfaces/DriverInterface.h"

#include "../../Header Files/UserInterface/Utils/PrintModel.h"

// меню заказов  
void orderMenu() {

std::map<int, std::function<void()>> menuOrder;

menuOrder[1] = addNewOrderInterface;

menuOrder[2] = getOrderInterface;

menuOrder[3] = updateOrderInterface;

menuOrder[4] = deleteOrderInterface;

menuOrder[5] = menu;

while (true) {

std::cout << "--------------------------\n"

<< "Команды:\n"

<< "Создать новый заказ - 1\n"

<< "Получить заказ - 2\n"

<< "Обновить заказ - 3\n"

<< "Удалить заказ - 4\n"

<< "Вернуться назад - 5\n"

<< "Введите команду: ";

std::string input; std::cin >> input;

try {

int command = stoi(input);

if (menuOrder.count(command))

menuOrder[command]();

else

std::cout << "Неверный ввод, попробуйте еще раз!" << std::endl;

}

catch (const std::exception& e) {

std::cout << "Ошибка: " << e.what() << std::endl;

}

}

}

// меню клиентов  
void clientMenu() {

std::map<int, std::function<void()>> menuClient;

menuClient[1] = addNewClientInterface;

menuClient[2] = getClientInterface;

menuClient[3] = updateClientInterface;

menuClient[4] = deleteClientInterface;

menuClient[5] = menu;

while (true) {

std::cout << "--------------------------\n"

<< "Команды:\n"

<< "Добавить нового клиента - 1\n"

<< "Получить клиента - 2\n"

<< "Обновить клиента - 3\n"

<< "Удалить клиента - 4\n"

<< "Вернуться назад - 5\n"

<< "Введите команду: ";

std::string input; std::cin >> input;

try {

int command = stoi(input);

if (menuClient.count(command))

menuClient[command]();

else

std::cout << "Неверный ввод, попробуйте еще раз!" << std::endl;

}

catch (const std::exception& e) {

std::cout << "Ошибка: " << e.what() << std::endl;

}

}

}

// меню водителей  
void driverMenu() {

std::map<int, std::function<void()>> menuDriver;

menuDriver[1] = addNewDriverInterface;

menuDriver[2] = getDriverInterface;

menuDriver[3] = updateDriverInterface;

menuDriver[4] = deleteDriverInterface;

menuDriver[5] = menu;

while (true) {

std::cout << "--------------------------\n"

<< "Команды:\n"

<< "Добавить нового водителя - 1\n"

<< "Получить водителя - 2\n"

<< "Обновить водителя - 3\n"

<< "Удалить водителя - 4\n"

<< "Вернуться назад - 5\n"

<< "Введите команду: ";

std::string input; std::cin >> input;

try {

int command = stoi(input);

if (menuDriver.count(command))

menuDriver[command]();

else

std::cout << "Неверный ввод, попробуйте еще раз!" << std::endl;

}

catch (const std::exception& e) {

std::cout << "Error: " << e.what() << std::endl;

}

}

}

// выход из программы  
void exitMenu() {

exit(0);

}

void menu() {

std::map<int, std::function<void()>> menuActions;

menuActions[1] = orderMenu;

menuActions[2] = clientMenu;

menuActions[3] = driverMenu;

menuActions[4] = exitMenu;

while (true) {

std::cout << "--------------------------\n"

<< "Команды:\n"

<< "База данных заказов - 1\n"

<< "База данных клиентов - 2\n"

<< "База данных водителей - 3\n"

<< "Выйти из программы - 4\n"

<< "Введите команду: ";

std::string input; std::cin >> input;

try {

int command = stoi(input);

if (menuActions.count(command))

menuActions[command]();

else

std::cout << "Неверный ввод, попробуйте еще раз!" << std::endl;

}

catch (const std::exception& e) {

std::cout << "Ошибка: " << e.what() << std::endl;

}

}

}

**Интерфейс заказов:**

// находится в OrderInterface.h  
#ifndef ORDERINTERFACE\_H

#define ORDERINTERFACE\_H

void addNewOrderInterface();

void getOrderInterface();

void updateOrderInterface();

void deleteOrderInterface();

#endif

// находится в OrderInterface.cpp  
#include <chrono>

#include <iostream>

#include <string>

#include "../../../Header Files/Models/Order.h"

#include "../../../Header Files/Models/Address.h"

#include "../../../Header Files/Services/OrderService.h"

#include "../../../Header Files/UserInterface/Interfaces/OrderInterface.h"

#include "../../../Header Files/UserInterface/Utils/PrintModel.h"

#include "../../../Header Files/UserInterface/Utils/OtherUtils.h"

// получение данных для добавления нового заказа  
void addNewOrderInterface() {

std::string input; std::cin.ignore();

std::cout << "--------------------------\n"

<< "Введите данные:" << std::endl;

Order order;

order.id = 0;

std::cout << "Введите id клиента: "; std::getline(std::cin, input); order.clientId = stoi(input);

std::cout << "Введите id водителя: "; std::getline(std::cin, input); order.driverId = stoi(input);

std::cout << "Введите адрес отправления:" << std::endl;

order.shippingAddress = enterAddress();

if (order.shippingAddress.city.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.shippingAddress.street.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.shippingAddress.house.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.shippingAddress.entrance.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.shippingAddress.apartment.empty()) throw std::runtime\_error("Invalid input!");

std::cout << "Введите адрес получателя:" << std::endl;

order.recipientAddress = enterAddress();

if (order.recipientAddress.city.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.recipientAddress.street.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.recipientAddress.house.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.recipientAddress.entrance.empty()) throw std::runtime\_error("Неверный ввод!");

if (order.recipientAddress.apartment.empty()) throw std::runtime\_error("Неверный ввод!");

std::cout << "Введите длину пути: "; std::getline(std::cin, input); order.routeLength = stoi(input);

order.date = std::chrono::system\_clock::to\_time\_t(std::chrono::system\_clock::now());

printOrder(createOrderService(order));

}

void getOrderInterface() {

std::string input; std::cin.ignore();

std::cout << "--------------------------\n"

<< "Введите id заказа: "; std::getline(std::cin, input);

printOrder(getOrderByIdService(stoi(input)));

}

// получение данных для обновления заказа  
void updateOrderInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< " Введите id заказа: "; std::getline(std::cin, input);

Order order;

order.id = std::stoi(input);

std::cout << "Command:\n"

<< "Обновить адрес отправления - 1\n"

<< "Обновить адрес получателя - 2\n"

<< "Введите команду: "; std::getline(std::cin, input);

if (std::stoi(input) == 1) {

std::cout << "Для каждого поля введите данные или enter, если данные не нужно обновлять: " << std::endl;

order.shippingAddress = enterAddress();

}

else if (std::stoi(input) == 2) {

std::cout << " Для каждого поля введите данные или enter, если данные не нужно обновлять: " << std::endl;

order.recipientAddress = enterAddress();

}

else {

throw std::runtime\_error("Неверный ввод!");

}

updateOrderService(order);

std::cout << "Адрес заказа успешно обновлен!" << std::endl;

}

// получение данных для удаления заказа  
void deleteOrderInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< "Введите id заказа: "; std::getline(std::cin, input);

deleteOrderService(stoi(input));

std::cout << "Заказ успешно обновлен!" << std::endl;

}

**Дополнительные функции для заказа:**

// находится в PrintModel.cpp  
// вывод информации о заказе

void printOrder(const Order& order) {

std::cout << "-----------------------------" << std::endl;

std::cout << "Id заказа: " << order.id << std::endl;

std::cout << "Id клиента: " << order.clientId << std::endl;

std::cout << "Id водителя: " << order.driverId << std::endl;

std::cout << "Адрес отправления: " << std::endl;

std::cout << "Город: " << order.shippingAddress.apartment << std::endl;

std::cout << "Улица: " << order.shippingAddress.street << std::endl;

std::cout << "Дом: " << order.shippingAddress.house << std::endl;

std::cout << "Подъезд: " << order.shippingAddress.entrance << std::endl;

std::cout << "Квартира: " << order.shippingAddress.apartment << std::endl;

std::cout << "Адрес получателя: " << std::endl;

std::cout << "Город: " << order.recipientAddress.apartment << std::endl;

std::cout << "Улица: " << order.recipientAddress.street << std::endl;

std::cout << "Дом: " << order.recipientAddress.house << std::endl;

std::cout << "Подъезд: " << order.recipientAddress.entrance << std::endl;

std::cout << "Квартира: " << order.recipientAddress.apartment << std::endl;

std::cout << "Длина дороги: " << order.routeLength << std::endl;

std::cout << "Время заказа: " << std::put\_time(std::localtime(&order.date), "%F %T") << std::endl;

}

// находится в OtherUtils.cpp  
// ввод информации о адресе  
Address enterAddress() {

Address address; std::string input;

std::cout << "Введите город: "; std::getline(std::cin, address.city);

std::cout << "Введите улицу: "; std::getline(std::cin, address.street);

std::cout << "Введите дом: "; std::getline(std::cin, address.house);

std::cout << "Введите подъезд: "; std::getline(std::cin, address.entrance);

std::cout << "Введите квартиру: "; std::getline(std::cin, address.apartment);

return address;

}

**Сервис заказов:**

// находится в OrderService.h  
#ifndef ORDERSERVICE\_H

#define ORDERSERVICE\_H

#include "../../Header Files/Models/Order.h"

Order createOrderService(Order);

Order getOrderByIdService(int);

void updateOrderService(Order);

void deleteOrderService(int);

#endif

// находится в OrderService.cpp  
#include <stdexcept>

#include "../../Header Files/Models/Address.h"

#include "../../Header Files/Models/Order.h"

#include "../../Header Files/Models/Client.h"

#include "../../Header Files/Models/Driver.h"

#include "../../Header Files/Services/OrderService.h"

#include "../../Header Files/Repository/OrderRepository.h"

#include "../../Header Files/Repository/ClientRepository.h"

#include "../../Header Files/Repository/DriverRepository.h"

// проверка ранее введенного заказа на валидность данных и передача этих данных

// дальше для занесения в базу данных

Order createOrderService(Order order) {

if (!checkByIdClientInDataBase(order.clientId))

throw std::runtime\_error("Пользователь не существует!");

if (!checkByIdDriverInDataBase(order.driverId))

throw std::runtime\_error("Водитель не существует!");

order.id = getLastOrderId() + 1;

Client client = getClientById(order.clientId);

client.orderHistory.push\_back(order.id);

updateClient(client);

Driver driver = getDriverById(order.driverId);

driver.orderHistory.push\_back(order.id);

updateDriver(driver);

return saveOrder(order);

}

// проверка на существование введенного заказа, чтобы получить из базы данных  
Order getOrderByIdService(int id) {

if (!checkByIdOrderInDataBase(id))

throw std::runtime\_error("The order does not exist!");

return getOrderById(id);

}

// проверка на валидность данных и обновление адреса  
void updateOrderService(Order order) {

if (!checkByIdOrderInDataBase(order.id))

throw std::runtime\_error("The order does not exist!");

Order updateOrderInf = getOrderById(order.id);

if (!order.shippingAddress.city.empty()) updateOrderInf.shippingAddress.city = order.shippingAddress.city;

if (!order.shippingAddress.street.empty()) updateOrderInf.shippingAddress.street = order.shippingAddress.street;

if (!order.shippingAddress.house.empty()) updateOrderInf.shippingAddress.house = order.shippingAddress.house;

if (!order.shippingAddress.entrance.empty()) updateOrderInf.shippingAddress.entrance = order.shippingAddress.entrance;

if (!order.shippingAddress.apartment.empty()) updateOrderInf.shippingAddress.apartment = order.shippingAddress.apartment;

if (!order.recipientAddress.city.empty()) updateOrderInf.recipientAddress.city = order.recipientAddress.city;

if (!order.recipientAddress.street.empty()) updateOrderInf.recipientAddress.street = order.recipientAddress.street;

if (!order.recipientAddress.house.empty()) updateOrderInf.recipientAddress.house = order.recipientAddress.house;

if (!order.recipientAddress.entrance.empty()) updateOrderInf.recipientAddress.entrance = order.recipientAddress.entrance;

if (!order.recipientAddress.apartment.empty()) updateOrderInf.recipientAddress.apartment = order.recipientAddress.apartment;

updateOrder(updateOrderInf);

}

// проверка на валидность данных, обновление истории клиента и водителя  
// и дальнейшее удаление заказа по id  
void deleteOrderService(int id) {

if (!checkByIdOrderInDataBase(id))

throw std::runtime\_error("The order does not exist!");

Order order = getOrderById(id);

Client client = getClientById(order.clientId);

for (int i = 0; i < client.orderHistory.size(); i++) {

if (client.orderHistory[i] == id) {

client.orderHistory.erase(client.orderHistory.begin() + i);

break;

}

}

updateClient(client);

Driver driver = getDriverById(order.driverId);

for (int i = 0; i < driver.orderHistory.size(); i++) {

if (driver.orderHistory[i] == id) {

driver.orderHistory.erase(driver.orderHistory.begin() + i);

break;

}

}

updateDriver(driver);

deleteOrderById(id);

}

**Взаимодействие с базой данных заказов:**

// находится в OrderRepository.h  
#ifndef ORDERREPOSITORY\_H

#define ORDERREPOSITORY\_H

#include "../../Header Files/Models/Order.h"

Order saveOrder(Order);

Order getOrderById(int);

void updateOrder(Order);

void deleteOrderById(int);

bool checkByIdOrderInDataBase(int);

int getLastOrderId();

#endif

// находится в OrderRepository.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

#include <chrono>

#include <fstream>

#include <iomanip>

#include <sstream>

#include <string>

#include <time.h>

#include "../../Header Files/config.h"

#include "../../Header Files/Models/Address.h"

#include "../../Header Files/Models/Order.h"

#include "../../Header Files/Models/Client.h"

#include "../../Header Files/Models/Driver.h"

#include "../../Header Files/Repository/OrderRepository.h"

#include "../../Header Files/Repository/ClientRepository.h"

#include "../../Header Files/Repository/DriverRepository.h"

#include "../../Header Files/Repository/FileParser/FileParser.h"

// сохранение информации о заказе в базе данных  
// добавление происходит в конец файла  
Order saveOrder(Order order) {

std::ofstream ordersFile(ORDERFILE, std::ios::app);

if (!ordersFile)

throw std::runtime\_error("Ошибка в OrderRepository: функция saveOrder - orderFile не найден!");

char buff[20];

strftime(buff, 20, "%F %T", localtime(&order.date));

std::string data =

std::to\_string(order.id) + '\n' +

std::to\_string(order.clientId) + '\n' +

std::to\_string(order.driverId) + '\n' +

order.shippingAddress.city + '\n' +

order.shippingAddress.street + '\n' +

order.shippingAddress.house + '\n' +

order.shippingAddress.entrance + '\n' +

order.shippingAddress.apartment + '\n' +

order.recipientAddress.city + '\n' +

order.recipientAddress.street + '\n' +

order.recipientAddress.house + '\n' +

order.recipientAddress.entrance + '\n' +

order.recipientAddress.apartment + '\n' +

std::to\_string(order.routeLength) + '\n' +

std::string(buff) + '\n' +

"~~~\n";

ordersFile << data;

ordersFile.close();

return order;

}

// получение заказа по id  
// нужно пройтись по всему файлу и искать id  
// когда заказ будет найден, нужно получить их и записать в структуру для удобства // работы с ней  
Order getOrderById(int id) {

std::ifstream ordersFile(ORDERFILE, std::ios::in);

if (!ordersFile)

throw std::runtime\_error("Ошибка в OrderRepository: функция getOrderById - orderFile не найден!");

std::string line;

while (std::getline(ordersFile, line)) {

if (std::stoi(line) == id) {

Order order;

order.id = stoi(line);

std::getline(ordersFile, line); order.clientId = std::stoi(line);

std::getline(ordersFile, line); order.driverId = std::stoi(line);

std::getline(ordersFile, order.shippingAddress.city);

std::getline(ordersFile, order.shippingAddress.street);

std::getline(ordersFile, order.shippingAddress.house);

std::getline(ordersFile, order.shippingAddress.entrance);

std::getline(ordersFile, order.shippingAddress.apartment);

std::getline(ordersFile, order.recipientAddress.city);

std::getline(ordersFile, order.recipientAddress.street);

std::getline(ordersFile, order.recipientAddress.house);

std::getline(ordersFile, order.recipientAddress.entrance);

std::getline(ordersFile, order.recipientAddress.apartment);

std::getline(ordersFile, line); order.routeLength = std::stoi(line);

std::getline(ordersFile, line);

// первый способ превращения даты из строки в time\_t

tm tm = {};

std::istringstream ss(line);

ss >> std::get\_time(&tm, "%Y-%m-%d %H:%M:%S");

order.date = mktime(&tm);

// второй способ превращения даты из строки в time\_t

/\*std::istringstream ss(line);

std::chrono::system\_clock::time\_point tp;

ss >> std::chrono::parse("%F %T", tp);

order.date = std::chrono::system\_clock::to\_time\_t(tp);\*/

ordersFile.close();

return order;

}

for (int i = 0; (i < 15) && !ordersFile.eof(); i++)

std::getline(ordersFile, line);

}

}

// обновление информации о заказе

// получение всей базы данных, нахождение нужного заказа, обновление информации и

// дальнейшего сохранение в базе данных

void updateOrder(Order order) {

std::ifstream ordersFile(ORDERFILE, std::ios::in);

if (!ordersFile)

throw std::runtime\_error("Ошибка в OrderRepository: функция updateOrder - orderFile не найден!");

std::vector<std::string> tempData = getAllDataFromFile(ordersFile);

ordersFile.close();

for (int i = 0; i < tempData.size(); i += 16) {

if (stoi(tempData[i]) == order.id) {

tempData[i + 0] = std::to\_string(order.id);

tempData[i + 1] = std::to\_string(order.clientId);

tempData[i + 2] = std::to\_string(order.driverId);

tempData[i + 3] = order.shippingAddress.city;

tempData[i + 4] = order.shippingAddress.street;

tempData[i + 5] = order.shippingAddress.house;

tempData[i + 6] = order.shippingAddress.entrance;

tempData[i + 7] = order.shippingAddress.apartment;

tempData[i + 8] = order.recipientAddress.city;

tempData[i + 9] = order.recipientAddress.street;

tempData[i + 10] = order.recipientAddress.house;

tempData[i + 11] = order.recipientAddress.entrance;

tempData[i + 12] = order.recipientAddress.apartment;

tempData[i + 13] = std::to\_string(order.routeLength);

char buff[20];

strftime(buff, 20, "%F %T", localtime(&order.date));

tempData[i + 14] = std::string(buff);

break;

}

}

std::ofstream ordersOutFile(ORDERFILE, std::ios::out);

if (!ordersOutFile)

throw std::runtime\_error("Ошибка в OrderRepository: фукнция updateOrder - orderFile не найден!");

writeDataInFile(ordersOutFile, tempData);

ordersOutFile.close();

}

// удаление заказа из базы данных по id  
// получение всех данных из базы данных, удаление заказа, запись обновленных данных

void deleteOrderById(int id) {

std::ifstream ordersFile(ORDERFILE, std::ios::in);

if (!ordersFile)

throw std::runtime\_error("Ошибка в OrderRepository: функция deleteOrderById - orderFile не найден!");

std::vector<std::string> tempData = getAllDataFromFile(ordersFile);

ordersFile.close();

for (int i = 0; i < tempData.size(); i += 16) {

if (stoi(tempData[i]) == id) {

for (int j = 0; j < 16; j++)

tempData.erase(tempData.begin() + i);

break;

}

}

std::ofstream driversOutFile(ORDERFILE, std::ios::out);

if (!driversOutFile)

throw std::runtime\_error("Ошибка в OrderRepository: функция deleteOrderById - clientFile не найден!");

writeDataInFile(driversOutFile, tempData);

driversOutFile.close();

}

// проверка существования заказа в базе данных  
// проход по всей базе данных для поиска переданного id  
bool checkByIdOrderInDataBase(int id) {

std::ifstream ordersFile(ORDERFILE, std::ios::in);

if (!ordersFile)

throw std::runtime\_error("Ошибка в OrderRepository: функция checkByIdOrderInDataBase - orderFile не найден!");

try {

bool answer = parsFieldInFile(ordersFile, id, 15);

ordersFile.close();

return answer;

}

catch (const std::exception& e) {

ordersFile.close();

throw std::runtime\_error("Ошибка в OrderRepository: функция checkByIdOrderInDataBase – ошибка обработки! Error: " + std::string(e.what()));

}

}

// получение id последнего заказа

int getLastOrderId() {

std::ifstream ordersFile(ORDERFILE, std::ios::in);

if (!ordersFile)

throw std::runtime\_error("Ошибка в OrderRepository: функция getLastOrderId - orderFile не найден!");

try {

std::string line; int lastId = 0, i = 0;

while (std::getline(ordersFile, line)) {

i++;

if (i == 1)

lastId = stoi(line);

if (i == 16)

i = 0;

}

return lastId;

}

catch (const std::exception& e) {

ordersFile.close();

throw std::runtime\_error("Ошибка в OrderRepository: функция getLastOrderId - orderFile не найден!" + std::string(e.what()));

}

}

**Интерфейс клиента:**

// расположен в ClientInterface.h  
#ifndef CLIENTINTERFACE\_H

#define CLIENTINTERFACE\_H

void addNewClientInterface();

void getClientInterface();

void updateClientInterface();

void deleteClientInterface();

#endif  
  
// расположен в ClientInterface.cpp  
#include <iostream>

#include <vector>

#include <string>

#include "../../../Header Files/Models/Order.h"

#include "../../../Header Files/Models/Client.h"

#include "../../../Header Files/Services/OrderService.h"

#include "../../../Header Files/Services/ClientService.h"

#include "../../../Header Files/UserInterface/Interfaces/ClientInterface.h"

#include "../../../Header Files/UserInterface/Utils/PrintModel.h"

#include "../../../Header Files/UserInterface/Utils/OtherUtils.h"

// ввод данных для добавления клиента в базу данных  
void addNewClientInterface() {

std::string input; std::cin.ignore();

std::cout << "--------------------------\n"

<< "Введите данные:" << std::endl;

Client client;

client.id = 0;

std::cout << "Введите уникальное имя: "; std::getline(std::cin, client.clientName);

if (client.clientName.empty()) throw std::runtime\_error("Ввод неверный!");

std::cout << "Введите имя: "; std::getline(std::cin, client.firstName);

if (client.firstName.empty()) throw std::runtime\_error("Ввод неверный!");

std::cout << "Введите фамилию: "; std::getline(std::cin, client.lastName);

if (client.lastName.empty()) throw std::runtime\_error("Ввод неверный!");

std::cout << "Введите день рождения:" << std::endl;

std::cout << "Введите день: "; std::getline(std::cin, input);

int day = stoi(input); (day > 0 || day <= 31) ? client.dateOfBirth[0] = day : throw std::runtime\_error("День неверен!");

std::cout << "Введите месяц: "; std::getline(std::cin, input);

int month = stoi(input); (month > 0 || month <= 31) ? client.dateOfBirth[1] = month : throw std::runtime\_error("Месяц неверен!");

std::cout << "Введите год: "; std::getline(std::cin, input);

int year = stoi(input); (year > 1925 || year <= 2025) ? client.dateOfBirth[2] = year : throw std::runtime\_error("Год неверен!");

client.orderHistory = {};

printClient(createClientService(client));

}

// ввод данных для получения информации о клиенте  
void getClientInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< "Команды:\n"

<< "Получить клиента с помощью id - 1\n"

<< "Получить клиента с помощью userName - 2\n"

<< "Получение информации о всех заказах с помощью id клиента - 3\n"

<< "Введите команду: "; std::getline(std::cin, input);

if (stoi(input) == 1) {

std::cout << "Введите id: "; std::getline(std::cin,input);

printClient(getClientService(stoi(input)));

}

else if (stoi(input) == 2) {

std::cout << "Ввдите clientName: "; std::getline(std::cin, input);

printClient(getClientService(input));

}

else if (stoi(input) == 3) {

std::cout << "Введите id: "; std::getline(std::cin, input);

Client answerClient = getClientService(stoi(input));

for (int i = 0; i < answerClient.orderHistory.size(); i++) {

printOrder(getOrderByIdService(answerClient.orderHistory[i]));

}

}

else {

throw std::runtime\_error("Неверный ввод!");

}

}

// ввод данных для обновления клиента  
void updateClientInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< "Команды:\n"

<< "Обновить с помощью id - 1\n"

<< "Обновить с помощью userName - 2\n"

<< "Введите команду: "; std::getline(std::cin, input);

Client client;

if (stoi(input) == 1) {

std::cout << "Введите id: "; std::getline(std::cin, input); client.id = stoi(input);

std::cout << "Введите userName(or enter to not change): "; std::getline(std::cin, input); client.clientName = input;

}

else if (stoi(input) == 2) {

std::cout << "Введите clientName: "; std::getline(std::cin, input);

client.id = 0; client.clientName = input;

}

else {

throw std::runtime\_error("Неверный ввод!");

}

updateClientService(enterClient(client));

std::cout << "Клиент успешно обновлен!" << std::endl;

}

// ввод данных для удаления клиента  
void deleteClientInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< "Команды:\n"

<< "Удалить с помощью id - 1\n"

<< "Удалить с помощью clientName - 2\n"

<< "Введите команду: "; std::getline(std::cin, input);

if (stoi(input) == 1) {

std::cout << "Введите id: "; std::getline(std::cin, input);

deleteClientService(stoi(input));

}

else if (stoi(input) == 2) {

std::cout << "Введите clientName: "; std::getline(std::cin, input);

deleteClientService(input);

}

else {

throw std::runtime\_error("Неверный ввод!");

}

std::cout << "Клиент успешно удален!" << std::endl;

}

**Дополнительные функции для клиента:**

// находится в PrintModel.cpp  
// вывод информации о клиенте

void printClient(const Client& client) {

std::cout << "-----------------------------" << std::endl;

std::cout << "Id: " << client.id << std::endl;

std::cout << "Уникальное имя: " << client.clientName << std::endl;

std::cout << "Имя: " << client.firstName << std::endl;

std::cout << "Фамилия: " << client.lastName << std::endl;

std::cout << "День рождения: ";

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

std::cout << client.dateOfBirth[i] << (i < 2 ? "-" : "");

std::cout << std::endl;

std::cout << "История заказов: ";

for (int i = 0; i < client.orderHistory.size(); i++)

std::cout << client.orderHistory[i] << (i < client.orderHistory.size()-1 ? "," : "");

std::cout << std::endl;

}

// находится в OtherUtils.cpp  
// ввод информации для обновления клиента  
Client enterClient(Client client) {

std::string input;

std::cout << "Введите имя(или enter чтобы не изменять): "; std::getline(std::cin, input); client.firstName = input;

std::cout << "Введите фамилию(или enter чтобы не изменять): "; std::getline(std::cin, input); client.lastName = input;

std::cout << "Введите день рождение(или enter чтобы не изменять): " << std::endl;

std::cout << "Введите день(или enter чтобы не изменять): "; std::getline(std::cin, input);

if (!input.empty()) {

int day = std::stoi(input);

(day > 0 || day <= 31) ? client.dateOfBirth[0] = day : throw std::runtime\_error("Неверная дата!");

}

else {

client.dateOfBirth[0] = 0;

}

std::cout << "Введите месяц(или enter чтобы не изменять): "; std::getline(std::cin, input);

if (!input.empty()) {

int month = std::stoi(input);

(month > 0 || month <= 12) ? client.dateOfBirth[1] = month : throw std::runtime\_error("Неверная дата!");

}

else {

client.dateOfBirth[1] = 0;

}

std::cout << "Введите год(или enter чтобы не изменять): "; std::getline(std::cin, input);

if (!input.empty()) {

int year = std::stoi(input);

(year > 1925 || year <= 2025) ? client.dateOfBirth[2] = year : throw std::runtime\_error("Неверная дата!");

}

else {

client.dateOfBirth[2] = 0;

}

return client;

}

**Сервис клиента:**// находится в ClientService.h  
#ifndef CLIENTSERVICE\_H

#define CLIENTSERVICE\_H

#include "../../Header Files/Models/Client.h"

Client createClientService(Client);

Client getClientService(int);

Client getClientService(std::string);

void updateClientService(Client);

void deleteClientService(int);

void deleteClientService(std::string);

#endif  
  
// находится в ClientService.cpp  
#include <stdexcept>

#include <vector>

#include "../../Header Files/Models/Client.h"

#include "../../Header Files/Services/ClientService.h"

#include "../../Header Files/Repository/ClientRepository.h"

// проверка клиента на существование в базе данных  
// получение последнего id и присвоение клиенту id + 1  
// дальнейшее сохранение а базе данных  
Client createClientService(Client client) {

if (checkByClientNameClientInDataBase(client.clientName))

throw std::runtime\_error("Это clientName уже занято!");

client.id = getLastClientId() + 1;

return saveClient(client);

}

// получение клиента по id

// проверка на существование клиента в базе данных  
// дальнейшее получение клиента по id из базы данных  
Client getClientService(int id) {

if (!checkByIdClientInDataBase(id))

throw std::runtime\_error("Клиент не существует!");

return getClientById(id);

}

// получение клиента по уникальному имени  
// проверка на существование клиента в базе данных

// дальнейшее получение клиента по clientName из базы данных  
Client getClientService(std::string clientName) {

if (!checkByClientNameClientInDataBase(clientName))

throw std::runtime\_error("The client does not exist!");

return getClientByClientName(clientName);

}

­­­­­­­// обновление информации о клиенте  
// проверка на существование клиента в базе данных  
// если данные введены, то они обновляются  
// обновление данных в базе данных  
void updateClientService(Client client) {

Client updateClientInf;

if (client.id != 0) {

if (!checkByIdClientInDataBase(client.id))

throw std::runtime\_error("Клиент не существует!");

updateClientInf = getClientById(client.id);

}

else {

if (!checkByClientNameClientInDataBase(client.clientName))

throw std::runtime\_error("Клиент не существует!");

updateClientInf = getClientByClientName(client.clientName);

}

if (!client.clientName.empty()) updateClientInf.clientName = client.clientName;

if (!client.firstName.empty()) updateClientInf.firstName = client.firstName;

if (!client.lastName.empty()) updateClientInf.lastName = client.lastName;

if (client.dateOfBirth[0] != 0) updateClientInf.dateOfBirth[0] = client.dateOfBirth[0];

if (client.dateOfBirth[1] != 0) updateClientInf.dateOfBirth[1] = client.dateOfBirth[1];

if (client.dateOfBirth[2] != 0) updateClientInf.dateOfBirth[2] = client.dateOfBirth[2];

if (!client.orderHistory.empty()) {

for (int i = 0; i < client.orderHistory.size(); i++) {

updateClientInf.orderHistory.push\_back(client.orderHistory[i]);

}

}

updateClient(updateClientInf);

}

// удаление клиента по id  
void deleteClientService(int id) {

if (!checkByIdClientInDataBase(id))

throw std::runtime\_error("Клиент не существует!");

deleteClientById(id);

}

// удаление клиента по clientName  
void deleteClientService(std::string clientName) {

Client client = getClientByClientName(clientName);

if (!checkByIdClientInDataBase(client.id))

throw std::runtime\_error("Клиент не существует!");

deleteClientById(client.id);

}

**Взаимодействие с базой данных клиентов:**

// находится в ClientRepository.h  
#ifndef CLIENTREPOSITORY\_H

#define CLIENTREPOSITORY\_H

#include "../../Header Files/Models/Client.h"

Client saveClient(Client);

Client getClientById(int);

Client getClientByClientName(std::string);

void updateClient(Client);

void deleteClientById(int);

bool checkByIdClientInDataBase(int);

void deleteClientByClientName(std::string);

bool checkByClientNameClientInDataBase(std::string);

int getLastClientId();

#endif  
  
  
// находится в ClientRepository.cpp  
#include <fstream>

#include <sstream>

#include "../../Header Files/config.h"

#include "../../Header Files/Models/Client.h"

#include "../../Header Files/Repository/ClientRepository.h"

#include "../../Header Files/Repository/FileParser/FileParser.h"

// сохранения клиента в базе данных  
Client saveClient(Client client) {

std::ofstream clientsFile(CLIENTFILE, std::ios::app);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция saveClient - clientFile не найден!");

std::stringstream orderHistory;

std::copy(client.orderHistory.begin(), client.orderHistory.end(), std::ostream\_iterator<int>(orderHistory, ","));

std::string data =

std::to\_string(client.id) + '\n' +

client.clientName + '\n' +

client.firstName + '\n' +

client.lastName + '\n' +

(client.dateOfBirth[0] < 10 ? "0" : "") + std::to\_string(client.dateOfBirth[0]) +

(client.dateOfBirth[1] < 10 ? "0" : "") + std::to\_string(client.dateOfBirth[1]) +

std::to\_string(client.dateOfBirth[2]) + '\n' +

orderHistory.str() + '\n' +

"~~~\n";

clientsFile << data;

clientsFile.close();

return client;

}

// получение клиента с помощь id  
Client getClientById(int id) {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция getClientById - clientFile не найден!");

std::string line; int currentLine = 0;

while (std::getline(clientsFile, line)) {

currentLine++;

if (std::stoi(line) == id) {

Client client;

client.id = stoi(line);

std::getline(clientsFile, client.clientName);

std::getline(clientsFile, client.firstName);

std::getline(clientsFile, client.lastName);

std::getline(clientsFile, line);

client.dateOfBirth[0] = std::stoi(line.substr(0, 2));

client.dateOfBirth[1] = std::stoi(line.substr(2, 2));

client.dateOfBirth[2] = std::stoi(line.substr(4, 4));

std::getline(clientsFile, line);

std::stringstream ss(line);

std::string number;

while (std::getline(ss, number, ',')) {

if (!number.empty())

client.orderHistory.push\_back(stoi(number));

}

clientsFile.close();

return client;

}

for (int i = 0; (i < 6) && !clientsFile.eof(); i++)

std::getline(clientsFile, line);

}

}

// получение клиента с помощь clientName

Client getClientByClientName(std::string clientName) {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция getClientByClientName - clientFile не найден!");

std::string line; int currentLine = 0;

std::getline(clientsFile, line);

int currentId = stoi(line);

while (std::getline(clientsFile, line)) {

currentLine++;

if (line == clientName) {

Client client;

client.id = currentId;

client.clientName = line;

std::getline(clientsFile, client.firstName);

std::getline(clientsFile, client.lastName);

std::getline(clientsFile, line);

client.dateOfBirth[0] = std::stoi(line.substr(0, 2));

client.dateOfBirth[1] = std::stoi(line.substr(2, 2));

client.dateOfBirth[2] = std::stoi(line.substr(4, 4));

std::getline(clientsFile, line);

std::stringstream ss(line);

std::string number;

while (std::getline(ss, number, ',')) {

if (!number.empty())

client.orderHistory.push\_back(stoi(number));

}

clientsFile.close();

return client;

}

for (int i = 0; (i < 6) && !clientsFile.eof(); i++) {

std::getline(clientsFile, line);

if (i == 5) currentId = stoi(line);

}

}

clientsFile.close();

throw std::runtime\_error("Ошибка в ClientRepository: функция getClientByClientName – Клиент не существует!");

}

// обновление информации у клиента в базе данных  
void updateClient(Client client) {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция updateClient - clientFile не найден!");

std::vector<std::string> tempData = getAllDataFromFile(clientsFile);

clientsFile.close();

for (int i = 0; i < tempData.size(); i += 7) {

if (stoi(tempData[i]) == client.id) {

tempData[i + 0] = std::to\_string(client.id);

tempData[i + 1] = client.clientName;

tempData[i + 2] = client.firstName;

tempData[i + 3] = client.lastName;

tempData[i + 4] = (client.dateOfBirth[0] < 10 ? "0" : "") + std::to\_string(client.dateOfBirth[0]) +

(client.dateOfBirth[1] < 10 ? "0" : "") + std::to\_string(client.dateOfBirth[1]) +

std::to\_string(client.dateOfBirth[2]);

std::stringstream result;

std::copy(client.orderHistory.begin(), client.orderHistory.end(), std::ostream\_iterator<int>(result, ","));

tempData[i + 5] = result.str();

break;

}

}

std::ofstream clientsOutFile(CLIENTFILE, std::ios::out);

if (!clientsOutFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция updateClient - clientFile не найден!");

writeDataInFile(clientsOutFile, tempData);

clientsOutFile.close();

}

// удаление клиента из базы данных с помощью id  
void deleteClientById(int id) {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция deleteClientById - clientFile не найден!");

std::vector<std::string> tempData = getAllDataFromFile(clientsFile);

clientsFile.close();

for (int i = 0; i < tempData.size(); i+=7) {

if (stoi(tempData[i]) == id) {

for (int j = 0; j < 7; j++)

tempData.erase(tempData.begin()+i);

break;

}

}

std::ofstream clientsOutFile(CLIENTFILE, std::ios::out);

if (!clientsOutFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция deleteClientById - clientFile не найден!");

writeDataInFile(clientsOutFile, tempData);

clientsOutFile.close();

}

// удаление клиента из базы данных с помощью clientName  
void deleteClientByClientName(std::string clientName) {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция deleteClientById - clientFile не найден!");

std::vector<std::string> tempData = getAllDataFromFile(clientsFile);

clientsFile.close();

for (int i = 1; i < tempData.size(); i += 7) {

if (tempData[i] == clientName) {

for (int j = 0; j < 7; j++)

tempData.erase(tempData.begin() + i - 1);

break;

}

}

std::ofstream clientsOutFile(CLIENTFILE, std::ios::out);

if (!clientsOutFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция deleteClientById - clientFile не найден!");

writeDataInFile(clientsOutFile, tempData);

clientsOutFile.close();

}

// проверка на существование клиента в базе данных с помощью id  
bool checkByIdClientInDataBase(int id) {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция checkClientInDataBase - clientFile не найден!");

try {

bool answer = parsFieldInFile(clientsFile, id, 6);

clientsFile.close();

return answer;

}

catch (const std::exception& e) {

clientsFile.close();

throw std::runtime\_error("Ошибка в ClientRepository: функция checkByIdClientInDataBase – ошибка обработки! " + std::string(e.what()));

}

}

// проверка на существование клиента в базе данных с помощью clientName  
bool checkByClientNameClientInDataBase(std::string clientName) {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция checkByUserNameClientInDataBase - clientFile не найден!");

try {

bool answer = parsFieldInFile(clientsFile, clientName, 1, 6);

clientsFile.close();

return answer;

}

catch (const std::exception& e) {

clientsFile.close();

throw std::runtime\_error("Ошибка в ClientRepository: функция checkByClientNameClientInDataBase – ошибка обработки! Ошибка: " + std::string(e.what()));

}

}

// получение id последнего клиента

int getLastClientId() {

std::ifstream clientsFile(CLIENTFILE, std::ios::in);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientRepository: функция getLastClientId - clientFile не найден!");

try {

std::string line; int lastId = 0, i = 0;

while (std::getline(clientsFile, line)) {

i++;

if (i == 1)

lastId = stoi(line);

if (i == 7)

i = 0;

}

return lastId;

}

catch (const std::exception& e) {

clientsFile.close();

throw std::runtime\_error("Ошибка в ClientRepository: функция getLastClientId – " + std::string(e.what()));

}

}

**Интерфейс водителя:**

// находится в DriverInterface.h  
#ifndef DRIVERINTERFACE\_H

#define DRIVERINTERFACE\_H

void addNewDriverInterface();

void getDriverInterface();

void updateDriverInterface();

void deleteDriverInterface();

#endif

// находится в DriverInterface.cpp  
#include <iostream>

#include <vector>

#include <string>

#include "../../../Header Files/Models/Order.h"

#include "../../../Header Files/Models/Driver.h"

#include "../../../Header Files/Models/Car.h"

#include "../../../Header Files/Services/OrderService.h"

#include "../../../Header Files/Services/DriverService.h"

#include "../../../Header Files/UserInterface/Interfaces/DriverInterface.h"

#include "../../../Header Files/UserInterface/Utils/PrintModel.h"

#include "../../../Header Files/UserInterface/Utils/OtherUtils.h"

// ввод данных для добавления водителя  
void addNewDriverInterface() {

std::string input; std::cin.ignore();

std::cout << "--------------------------\n"

<< "Введите данные:" << std::endl;

Driver driver;

driver.id = 0;

std::cout << "Введите уникальное имя водителя: "; std::getline(std::cin, input);

!input.empty() ? driver.driverName = input : throw std::runtime\_error("Неверный ввод!");

std::cout << "Введите имя: "; std::getline(std::cin, input);

!input.empty() ? driver.firstName = input : throw std::runtime\_error("Неверный ввод!");

std::cout << "Введите фамилию: "; std::getline(std::cin, input);

!input.empty() ? driver.lastName = input : throw std::runtime\_error("Неверный ввод!");

std::cout << "Введите день рождения:" << std::endl;

std::cout << "Введите день: "; std::getline(std::cin, input);

int day = stoi(input); (day > 0 || day <= 31) ? driver.dateOfBirth[0] = day: throw std::runtime\_error("Неверная дата!");

std::cout << "Введите месяц: "; std::getline(std::cin, input);

int month = stoi(input); (month > 0 || month <= 12) ? driver.dateOfBirth[1] = month : throw std::runtime\_error("Неверная дата!");

std::cout << "Введите месяц: "; std::getline(std::cin, input);

int year = stoi(input); (year > 1925 || year <= 2025) ? driver.dateOfBirth[2] = year : throw std::runtime\_error("Неверная дата!");

std::cout << "Введите опыт вождения: "; std::getline(std::cin, input);

int drivingExperience = stoi(input); drivingExperience > 0 ? driver.drivingExperience = drivingExperience : throw std::runtime\_error("Неверный ввод!");

Car car;

std::cout << "Введите информацию о машине: " << std::endl;

std::cout << "Введите марку: "; std::getline(std::cin, driver.car.brand);

std::cout << "Введите модель: "; std::getline(std::cin, driver.car.model);

std::cout << "Введите класс: "; std::getline(std::cin, driver.car.classC);

std::cout << "Введите год:"; std::getline(std::cin, input); driver.car.year = stoi(input);

printDriver(createDriverService(driver));

}

// ввод информации для получения водителя  
void getDriverInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< "Команды:\n"

<< "Получить с помощью id - 1\n"

<< "Получить с помощью driverName - 2\n"

<< "Получить историю заказов с помощью id - 3\n"

<< "Введите команду: "; std::getline(std::cin, input);

if (stoi(input) == 1) {

std::cout << "Введите id: "; std::getline(std::cin, input);

printDriver(getDriverService(stoi(input)));

}

else if (stoi(input) == 2) {

std::cout << "Введите driverName: "; std::getline(std::cin, input);

printDriver(getDriverService(input));

}

else if (stoi(input) == 3) {

std::cout << "Введите id: "; std::getline(std::cin, input);

Driver answerDriver = getDriverService(stoi(input));

for (int i = 0; i < answerDriver.orderHistory.size(); i++) {

printOrder(getOrderByIdService(answerDriver.orderHistory[i]));

}

}

else {

throw std::runtime\_error("Неверный ввод!");

}

}

// Ввод информации для обновления водителя  
void updateDriverInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< "Команды:\n"

<< "Обновить с помощью id - 1\n"

<< "Обновить с помощью driverName - 2\n"

<< "Введите команду: "; std::getline(std::cin, input);

Driver driver;

if (stoi(input) == 1) {

std::cout << "Введите id: "; std::getline(std::cin, input);

driver.id = stoi(input);

}

else if (stoi(input) == 2) {

driver.id = 0;

}

else {

throw std::runtime\_error("Неверный ввод!");

}

updateDriverService(enterDriver(driver));

std::cout << "Водитель успешно обновлен!" << std::endl;

}

// ввод информации для удаления водителя  
void deleteDriverInterface() {

std::string input; std::cin.ignore();

std::cout << "-----------------------------\n"

<< "Команды:\n"

<< "Удалить с помощью id - 1\n"

<< "Удалить с помощью driverName - 2\n"

<< "Введите команду: "; std::getline(std::cin, input);

if (stoi(input) == 1) {

std::cout << "Введите id: "; std::getline(std::cin, input);

deleteDriverService(stoi(input));

}

else if (stoi(input) == 2) {

std::cout << "Введите clientName: "; std::getline(std::cin, input);

deleteDriverService(input);

}

else {

throw std::runtime\_error("Неверный ввод!");

}

std::cout << "Водитель успешно удален!" << std::endl;

}

**Дополнительные функции для водителя:**

// находится в PrintModel.cpp  
// вывод информации о водителе

void printDriver(const Driver& driver) {

std::cout << "-----------------------------" << std::endl;

std::cout << "Id: " << driver.id << std::endl;

std::cout << "Уникальное имя водителя: " << driver.driverName << std::endl;

std::cout << "Имя: " << driver.firstName << std::endl;

std::cout << "Фамилия: " << driver.lastName << std::endl;

std::cout << "День рождения: ";

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

std::cout << driver.dateOfBirth[i] << (i < 2 ? "-" : "");

std::cout << std::endl;

std::cout << "Опыт вождения: " << driver.drivingExperience << std::endl;

std::cout << "Автомобиль:" << std::endl;

std::cout << "Марка: " << driver.car.brand << std::endl;

std::cout << "Модель: " << driver.car.model << std::endl;

std::cout << "Класс: " << driver.car.classC << std::endl;

std::cout << "Год: " << driver.car.year << std::endl;

std::cout << "История заказов: ";

for (int i = 0; i < driver.orderHistory.size(); i++)

std::cout << driver.orderHistory[i] << (i < driver.orderHistory.size()-1 ? "," : "");

std::cout << std::endl;

}

// находится в OtherUtils.cpp  
// ввод информации для обновления водителя  
Driver enterDriver(Driver driver) {

std::string input;

if (driver.id == 0) std::cout << "Введите driverName: "; std::getline(std::cin, driver.driverName);

std::cout << "Введите имя(или enter чтобы не изменять): "; std::getline(std::cin, input); driver.firstName = input;

std::cout << "Введите фамилию(или enter чтобы не изменять): "; std::getline(std::cin, input); driver.lastName = input;

std::cout << "Введите день рождения(или enter чтобы не изменять): " << std::endl;

std::cout << "Введите день(или enter чтобы не изменять): "; std::getline(std::cin, input);

if (!input.empty()) {

int day = std::stoi(input);

(day > 0 || day <= 31) ? driver.dateOfBirth[0] = day : throw std::runtime\_error("Неверная дата!");

}

else {

driver.dateOfBirth[0] = 0;

}

std::cout << "Введите месяц(или enter чтобы не изменять): "; std::getline(std::cin, input);

if (!input.empty()) {

int month = std::stoi(input);

(month > 0 || month <= 31) ? driver.dateOfBirth[1] = month : throw std::runtime\_error("Неверная дата!");

}

else {

driver.dateOfBirth[0] = 0;

}

std::cout << "Введите год(или enter чтобы не изменять): "; std::getline(std::cin, input);

if (!input.empty()) {

int year = std::stoi(input);

(year > 1925 || year <= 2025) ? driver.dateOfBirth[2] = year : throw std::runtime\_error("Неверная дата!");

}

else {

driver.dateOfBirth[0] = 0;

}

std::cout << "Введите опыт вождения(или enter чтобы не изменять "; std::getline(std::cin, input);

!input.empty() ? driver.drivingExperience = stoi(input) : driver.drivingExperience = 0;

Car car;

std::cout << "Введите информацию о автомобили: " << std::endl;

std::cout << "Введите марку: "; std::getline(std::cin, car.brand);

std::cout << "Введите модель: "; std::getline(std::cin, car.model);

std::cout << "Введите класс: "; std::getline(std::cin, car.classC);

std::cout << "Введите год: "; std::getline(std::cin, input);

!input.empty() ? car.year = stoi(input) : car.year = 0;

driver.car = car;

return driver;

}

**Сервис водителя:**

// находится в DriverService.h  
#ifndef DRIVERSERVICE\_H

#define DRIVERSERVICE\_H

#include "../../Header Files/Models/Driver.h"

Driver createDriverService(Driver);

Driver getDriverService(int);

Driver getDriverService(std::string);

void updateDriverService(Driver);

void deleteDriverService(int);

void deleteDriverService(std::string);

#endif

// находится в DriverService.cpp  
#include <stdexcept>

#include <vector>

#include "../../Header Files/Services/DriverService.h"

#include "../../Header Files/Repository/DriverRepository.h"

// проверка существования водителя в базе данных  
// получение последнего id и добавление к нему + 1  
// сохранение в базе данных

Driver createDriverService(Driver driver) {

if (checkByIdDriverInDataBase(driver.id))

throw std::runtime\_error("Водитель уже существует!");

driver.id = getLastDriverId() + 1;

return saveDriver(driver);

}

// проверка существования водителя в базе данных   
// получение из базы данных с помощью id  
Driver getDriverService(int id) {

if (!checkByIdDriverInDataBase(id))

throw std::runtime\_error("Водитель не существует!");

return getDriverById(id);

}

// проверка существования водителя в базе данных

// получение из базы данных с помощью driverName

Driver getDriverService(std::string driverName) {

if (!checkByDriverNameInDataBase(driverName))

throw std::runtime\_error("Водитель не существует!");

return getDriverByDriverName(driverName);

}

// проверка существования водителя в базе данных  
// если данные введены, то обновление, иначе данные не обновляются  
// обновление в базе данных

void updateDriverService(Driver driver) {

Driver updateDriverInf;

if (driver.id != 0) {

if (!checkByIdDriverInDataBase(driver.id))

throw std::runtime\_error("Водитель не существует!");

updateDriverInf = getDriverById(driver.id);

}

else {

if (!checkByDriverNameInDataBase(driver.driverName))

throw std::runtime\_error("Водитель не существует!");

updateDriverInf = getDriverByDriverName(driver.driverName);

}

if (!driver.driverName.empty()) updateDriverInf.driverName = driver.driverName;

if (!driver.firstName.empty()) updateDriverInf.firstName = driver.firstName;

if (!driver.lastName.empty()) updateDriverInf.lastName = driver.lastName;

if (driver.dateOfBirth[0] != 0) updateDriverInf.dateOfBirth[0] = driver.dateOfBirth[0];

if (driver.dateOfBirth[1] != 0) updateDriverInf.dateOfBirth[1] = driver.dateOfBirth[1];

if (driver.dateOfBirth[2] != 0) updateDriverInf.dateOfBirth[2] = driver.dateOfBirth[2];

if (driver.drivingExperience != 0) updateDriverInf.drivingExperience = driver.drivingExperience;

if (!driver.car.brand.empty()) updateDriverInf.car.brand = driver.car.brand;

if (!driver.car.model.empty()) updateDriverInf.car.model = driver.car.model;

if (!driver.car.classC.empty()) updateDriverInf.car.classC = driver.car.classC;

if (driver.car.year != 0) updateDriverInf.car.year = driver.car.year;

if (!driver.orderHistory.empty()) {

for (int i = 0; i < driver.orderHistory.size(); i++) {

updateDriverInf.orderHistory.push\_back(driver.orderHistory[i]);

}

}

updateDriver(updateDriverInf);

}

// проверка существования водителя в базе данных  
// удаление в базе данных с помощью id

void deleteDriverService(int id) {

if (!checkByIdDriverInDataBase(id))

throw std::runtime\_error("Водитель не существует!");

deleteDriverById(id);

}

// проверка существования водителя в базе данных  
// удаление в базе данных с помощью driverName

void deleteDriverService(std::string driverName) {

if (!checkByDriverNameInDataBase(driverName))

throw std::runtime\_error("Водитель не существует!");

Driver driver = getDriverByDriverName(driverName);

deleteDriverById(driver.id);

}

**Взаимодействие с базой данных водителей:**

// находится в DriverRepository.h  
#ifndef DRIVERREPOSITORY\_H

#define DRIVERREPOSITORY\_H

Driver saveDriver(Driver);

Driver getDriverById(int);

Driver getDriverByDriverName(std::string);

void updateDriver(Driver);

void deleteDriverById(int);

bool checkByIdDriverInDataBase(int);

bool checkByDriverNameInDataBase(std::string);

int getLastDriverId();

#endif

// находится в DriverRepository.cpp  
#include <fstream>

#include <sstream>

#include <string>

#include "../../Header Files/config.h"

#include "../../Header Files/Models/Car.h"

#include "../../Header Files/Models/Driver.h"

#include "../../Header Files/Repository/ClientRepository.h"

#include "../../Header Files/Repository/FileParser/FileParser.h"

// сохранение водителя в базе данных  
Driver saveDriver(Driver driver) {

std::ofstream driversFile(DRIVERFILE, std::ios::app);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция saveDriver - driverFile не найден!");

std::stringstream orderHistory;

std::copy(driver.orderHistory.begin(), driver.orderHistory.end(), std::ostream\_iterator<int>(orderHistory, ","));

std::string data =

std::to\_string(driver.id) + '\n' +

driver.driverName + '\n' +

driver.firstName + '\n' +

driver.lastName + '\n' +

(driver.dateOfBirth[0] < 10 ? "0" : "") + std::to\_string(driver.dateOfBirth[0]) +

(driver.dateOfBirth[1] < 10 ? "0" : "") + std::to\_string(driver.dateOfBirth[1]) +

std::to\_string(driver.dateOfBirth[2]) + '\n' +

std::to\_string(driver.drivingExperience) + '\n' +

driver.car.brand + '\n' +

driver.car.model + '\n' +

driver.car.classC + '\n' +

std::to\_string(driver.car.year) + '\n' +

orderHistory.str() + '\n' +

"~~~\n";

driversFile << data;

driversFile.close();

return driver;

}

// получение водителя с помощью id  
Driver getDriverById(int id) {

std::ifstream driversFile(DRIVERFILE, std::ios::in);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция getDriver - driverFile не найден!");

std::string line; int currentLine = 0;

while (std::getline(driversFile, line)) {

currentLine++;

if (std::stoi(line) == id) {

Driver driver;

driver.id = stoi(line);

std::getline(driversFile, driver.driverName);

std::getline(driversFile, driver.firstName);

std::getline(driversFile, driver.lastName);

std::getline(driversFile, line);

driver.dateOfBirth[0] = std::stoi(line.substr(0, 2));

driver.dateOfBirth[1] = std::stoi(line.substr(2, 2));

driver.dateOfBirth[2] = std::stoi(line.substr(4, 4));

std::getline(driversFile, line);

driver.drivingExperience = stoi(line);

std::getline(driversFile, driver.car.brand);

std::getline(driversFile, driver.car.model);

std::getline(driversFile, driver.car.classC);

std::getline(driversFile, line);

driver.car.year = stoi(line);

std::getline(driversFile, line);

std::stringstream ss(line);

std::string number;

while (std::getline(ss, number, ',')) {

if (!number.empty())

driver.orderHistory.push\_back(stoi(number));

}

driversFile.close();

return driver;

}

for (int i = 0; (i < 11) && !driversFile.eof(); i++)

std::getline(driversFile, line);

}

driversFile.close();

throw std::runtime\_error("Ошибка в DriverRepository: функция getDriverById – Водитель на существует!");

}

// получение водителя с помощью driverName

Driver getDriverByDriverName(std::string driverName) {

std::ifstream driversFile(DRIVERFILE, std::ios::in);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция getDriverByDriverName - driverFile не найден!");

std::string line; int currentLine = 0;

std::getline(driversFile, line);

int currentId = stoi(line);

while (std::getline(driversFile, line)) {

currentLine++;

if (line == driverName) {

Driver driver;

driver.id = currentId;

driver.driverName = line;

std::getline(driversFile, driver.firstName);

std::getline(driversFile, driver.lastName);

std::getline(driversFile, line);

driver.dateOfBirth[0] = std::stoi(line.substr(0, 2));

driver.dateOfBirth[1] = std::stoi(line.substr(2, 2));

driver.dateOfBirth[2] = std::stoi(line.substr(4, 4));

std::getline(driversFile, line);

driver.drivingExperience = stoi(line);

std::getline(driversFile, driver.car.brand);

std::getline(driversFile, driver.car.model);

std::getline(driversFile, driver.car.classC);

std::getline(driversFile, line);

driver.car.year = stoi(line);

std::getline(driversFile, line);

std::stringstream ss(line);

std::string number;

while (std::getline(ss, number, ',')) {

if (!number.empty())

driver.orderHistory.push\_back(stoi(number));

}

driversFile.close();

return driver;

}

for (int i = 0; (i < 11) && !driversFile.eof(); i++) {

std::getline(driversFile, line);

if (i == 10) currentId = stoi(line);

}

}

driversFile.close();

throw std::runtime\_error("Ошибка в DriverRepository: функция getDriverByDriverName – Водитель на существует!");

}

// обновление водителя в базе данных  
void updateDriver(Driver driver) {

std::ifstream driversFile(DRIVERFILE, std::ios::in);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция updateClient - driverFile не найден!");

std::vector<std::string> tempData = getAllDataFromFile(driversFile);

driversFile.close();

for (int i = 0; i < tempData.size(); i += 12) {

if (stoi(tempData[i]) == driver.id) {

tempData[i + 0] = std::to\_string(driver.id);

tempData[i + 1] = driver.driverName;

tempData[i + 2] = driver.firstName;

tempData[i + 3] = driver.lastName;

tempData[i + 4] = (driver.dateOfBirth[0] < 10 ? "0" : "") + std::to\_string(driver.dateOfBirth[0]) +

(driver.dateOfBirth[1] < 10 ? "0" : "") + std::to\_string(driver.dateOfBirth[1]) +

std::to\_string(driver.dateOfBirth[2]);

tempData[i + 5] = std::to\_string(driver.drivingExperience);

tempData[i + 6] = driver.car.brand;

tempData[i + 7] = driver.car.model;

tempData[i + 8] = driver.car.classC;

tempData[i + 9] = std::to\_string(driver.car.year);

std::stringstream result;

std::copy(driver.orderHistory.begin(), driver.orderHistory.end(), std::ostream\_iterator<int>(result, ","));

tempData[i + 10] = result.str();

break;

}

}

std::ofstream driversOutFile(DRIVERFILE, std::ios::out);

if (!driversOutFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция updateDriver - driverFile не найден!");

writeDataInFile(driversOutFile, tempData);

driversOutFile.close();

}

// удаление водителя из базы данных с помощью id

void deleteDriverById(int id) {

std::ifstream driversFile(DRIVERFILE, std::ios::in);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция deleteDriverById - driverFile не найден!");

std::vector<std::string> tempData = getAllDataFromFile(driversFile);

driversFile.close();

for (int i = 0; i < tempData.size(); i += 12) {

if (stoi(tempData[i]) == id) {

for (int j = 0; j < 12; j++)

tempData.erase(tempData.begin() + i);

break;

}

}

std::ofstream driversOutFile(DRIVERFILE, std::ios::out);

if (!driversOutFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция deleteDriverById - driverFile не найден!");

writeDataInFile(driversOutFile, tempData);

driversOutFile.close();

}

// проверка существования водителя в базе данных с помощью id  
bool checkByIdDriverInDataBase(int id) {

std::ifstream driversFile(DRIVERFILE, std::ios::in);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция checkByIdClientInDataBase - driverFile не найден!");

try {

bool answer = parsFieldInFile(driversFile, id, 11);

driversFile.close();

return answer;

}

catch (const std::exception& e) {

driversFile.close();

throw std::runtime\_error("Ошибка в DriverRepository: функция checkByIdDriverInDataBase - ошибка обработки!" + std::string(e.what()));

}

}

// проверка существования водителя в базе данных с помощью driverName

bool checkByDriverNameInDataBase(std::string driverName) {

std::ifstream driversFile(DRIVERFILE, std::ios::in);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция checkByDriverNameInDataBase - driverFile не найден!");

try {

bool answer = parsFieldInFile(driversFile, driverName, 1, 11);

driversFile.close();

return answer;

}

catch (const std::exception& e) {

driversFile.close();

throw std::runtime\_error("Ошибка в DriverRepository: функция checkByIdDriverInDataBase – ошибка обработки!" + std::string(e.what()));

}

}

// получение id последнего водителя int getLastDriverId() {

std::ifstream driversFile(DRIVERFILE, std::ios::in);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverRepository: функция checkByDriverNameInDataBase - driverFile не найден!");

try {

std::string line; int lastId = 0, i = 0;

while (std::getline(driversFile, line)) {

i++;

if (i == 1)

lastId = stoi(line);

if (i == 12)

i = 0;

}

return lastId;

}

catch (const std::exception& e) {

driversFile.close();

throw std::runtime\_error("Ошибка в DriverRepository: функция getLastDriverId - " + std::string(e.what()));

}

}

**Дополнительные функции для взаимодействия с импровизированной базой данных:**

// находится в FileParser.h  
#ifndef FILEPARSER\_H

#define FILEPARSER\_H

#include <vector>

bool parsFieldInFile(std::ifstream&, int, int);

bool parsFieldInFile(std::ifstream&, std::string, int, int);

std::vector<std::string> getAllDataFromFile(std::ifstream&);

void writeDataInFile(std::ofstream&, std::vector<std::string>);

#endif  
  
// находится в FileParser.cpp  
#include <fstream>

#include <vector>

#include <string>

#include "../../../Header Files/Repository/FileParser/FileParser.h"

// поиск поля в базе данных  
bool parsFieldInFile(std::ifstream& file, int id, int jump) {

std::string line;

while (std::getline(file, line)) {

if (std::stoi(line) == id)

return true;

for (int i = 0; (i < jump) && !file.eof(); i++)

std::getline(file, line);

}

return false;

}

// поиск поля в базе данных

bool parsFieldInFile(std::ifstream& file, std::string str, int goToStartPosition, int jump) {

std::string line;

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

std::getline(file, line);

while (std::getline(file, line)) {

std::string currentUserName = line;

if (currentUserName == str)

return true;

for (int i = 0; (i < 11) && !file.eof(); i++)

std::getline(file, line);

}

return false;

}

// получение всей базы данных  
std::vector<std::string> getAllDataFromFile(std::ifstream& file) {

std::vector<std::string> tempData;

std::string line;

while (std::getline(file, line)) {

tempData.push\_back(line);

}

return tempData;

}

// записать информации в базу данных  
void writeDataInFile(std::ofstream& file, std::vector<std::string> tempData) {

for (int i = 0; i < tempData.size(); i++) {

file << tempData[i];

file << '\n';

}

}

**Функции генерации базы данных для тестов:**

// находится в OrderSeed.h  
#ifndef ORDERSEED\_H

#define ORDERSEED\_H

void deleteDataInOrderDataBase();

void initializationOrderDataBase();

#endif

// находится в OrderSeed.cpp  
#include <fstream>

#include "../../Header Files/config.h"

#include "../../Header Files/Models/Address.h"

#include "../../Header Files/Models/Order.h"

#include "../../Header Files/Models/Client.h"

#include "../../Header Files/Models/Driver.h"

#include "../../Header Files/InitData/OrderSeed.h"

#include "../../Header Files/Services/OrderService.h"

#include "../../Header Files/Services/ClientService.h"

#include "../../Header Files/Services/DriverService.h"

#include "../../Header Files/Repository/ClientRepository.h"

#include "../../Header Files/Repository/DriverRepository.h"

// удаление всей информации в базе данных заказов  
void deleteDataInOrderDataBase() {

std::ofstream ordersFile(ORDERFILE, std::ofstream::out | std::ofstream::trunc);

if (!ordersFile)

throw std::runtime\_error("Ошибка в OrderSeed: функция deleteDataInOrderDataBase - orderFile не найден!");

ordersFile.close();

}

// получение количества клиентов и водителей  
// генерация заказов в количестве – клиенты умноженные на водителей  
// для каждого заказа берется рандомный клиент и водитель

void initializationOrderDataBase() {

std::vector<Client> allClients;

std::vector<Driver> allDrivers;

int clintsLastId = getLastClientId();

int driversLastId = getLastDriverId();

std::time\_t lastTime = 1735689600;

for (int i = 1; i <= clintsLastId \* driversLastId; i++) {

int randomClient = 1 + rand() % clintsLastId;

int randomDriver = 1 + rand() % driversLastId;

Address shippingAddress = {

"testCityShippingAddress" + std::to\_string(i),

"testStreetShippingAddress" + std::to\_string(i),

"testHouseShippingAddress" + std::to\_string(i),

"testEntranceShippingAddress" + std::to\_string(i),

"testApartmentShippingAddress" + std::to\_string(i),

};

Address recipientAddress = {

"testCityRecipientAddress" + std::to\_string(i),

"testStreetRecipientAddress" + std::to\_string(i),

"testHouseRecipientAddress" + std::to\_string(i),

"testEntranceRecipientAddress" + std::to\_string(i),

"testApartmentRecipientAddress" + std::to\_string(i),

};

lastTime = (time\_t)(lastTime + (rand() \* rand()) % ((15 \* 24 \* 60 \* 60) + 1));

Order orderTest = {

i,

randomClient,

randomDriver,

shippingAddress,

recipientAddress,

0 + rand() % 300,

lastTime

};

createOrderService(orderTest);

}

}

// находится в ClientSeed.h   
#ifndef CLIENDSEED\_H

#define CLIENDSEED\_H

void deleteDataInClientDataBase();

void initializationClientDataBase();

#endif

// находится в ClientSeed.cpp   
#include <fstream>

#include <string>

#include "../../Header Files/config.h"

#include "../../Header Files/Models/Client.h"

#include "../../Header Files/InitData/ClientSeed.h"

#include "../../Header Files/Services/ClientService.h"

// удаление всей информации в базе данных клиентов

void deleteDataInClientDataBase() {

std::ofstream clientsFile(CLIENTFILE, std::ofstream::out | std::ofstream::trunc);

if (!clientsFile)

throw std::runtime\_error("Ошибка в ClientSeed: функция deleteDataInClientDataBase - clientFile не найден!");

clientsFile.close();

}

// генерация клиентов

void initializationClientDataBase() {

for (int i = 0; i < 10; i++) {

Client clientTest = {

i + 1,

"test" + std::to\_string(i+1),

"testFirstName" + std::to\_string(i+1),

"testLastName" + std::to\_string(i+1),

{0 + rand() % 32, 0 + rand() % 13, 2000 + rand() % 25},

{}

};

createClientService(clientTest);

}

}

// находится DriverSeed.h   
#ifndef DRIVERSEED\_H

#define DRIVERSEED\_H

void deleteDataInDriverDataBase();

void initializationDriverDataBase();

#endif

// находится в DriverSeed.cpp   
#include <fstream>

#include <string>

#include "../../Header Files/config.h"

#include "../../Header Files/Models/Car.h"

#include "../../Header Files/Models/Driver.h"

#include "../../Header Files/InitData/DriverSeed.h"

#include "../../Header Files/Services/DriverService.h"

// удаление всей информации в базе данных водителей

void deleteDataInDriverDataBase() {

std::ofstream driversFile(DRIVERFILE, std::ofstream::out | std::ofstream::trunc);

if (!driversFile)

throw std::runtime\_error("Ошибка в DriverSeed: функция deleteDataInDriverDataBase - driverFile не найден!");

driversFile.close();

}

// генерация водителей

void initializationDriverDataBase() {

for (int i = 0; i < 10; i++) {

Car car = {

"testBrand" + std::to\_string(i + 1),

"testModel" + std::to\_string(i + 1),

"testClass" + std::to\_string(i + 1),

2000 + rand() % 25

};

Driver driverTest = {

i + 1,

"testDriver" + std::to\_string(i + 1),

"testDriverFirstName" + std::to\_string(i + 1),

"testDriverLastName" + std::to\_string(i + 1),

{0 + rand() % 32, 0 + rand() % 13, 2000 + rand() % 25},

(2 + rand() % 12),

car,

{}

};

createDriverService(driverTest);

}

}

**Пример работы программы:**



