Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информационных технологий»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Основы алгоритмизации и программирования

К ЗАЩИТЕ ДОПУСТИТЬ

\_\_\_\_\_\_\_\_ Ковальчук А. М.

Отчёт

по учебной практике (ознакомительной)

по теме

«Заказ транспортных средств»

БГУИР КР 6-05-0611-05 106 ПЗ

Студент ФИО

( гр. )

Руководитель Ковальчук А.М.

МИНСК 2025

**СОДЕРЖАНИЕ**

УСЛОВИЕ ЗАДАНИЯ............................................................................................3

ВВЕДЕНИЕ..............................................................................................................4

1 ФУНКЦИОНАЛЬНОЕ ПРОЭКТИРОВАНИЕ..................................................5

1.1 Входные и выходные данные.................................................................5

2 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ.................................................7

2.1 Разработка схем алгоритмов..................................................................7

2.2 Разработка алгоритмов...........................................................................7

3 РЕЗУЛЬТАТ РАБОТЫ.........................................................................................9

ЗАКЛЮЧЕНИЕ.....................................................................................................12

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ............................................13

ПРИЛОЖЕНИЕ А.................................................................................................14

ПРИЛОЖЕНИЕ Б..................................................................................................15

ПРИЛОЖЕНИЕ В..................................................................................................16

ПРИЛОЖЕНИЕ Г..................................................................................................17

ПРИЛОЖЕНИЕ Д..................................................................................................19

ПРИЛОЖЕНИЕ Е..................................................................................................22

ПРИЛОЖЕНИЕ Ж.................................................................................................28

ПРИЛОЖЕНИЕ З...................................................................................................64

**УСЛОВИЕ ЗАДАНИЯ**

В файлах (бинарных или текстовых) хранятся сведения о транспортных средствах (грузовых и пассажирских). Написать программу для заказа транспортных средств для выполнения перевозок, с возможностью:

1. Заказа транспортного средства (грузового или пассажирского) на определённый день

2. Корректировки наличия транспортных средств

3. Загрузки из файла и выгрузки в файл

4. Редактирования и отмены бронирования

5. Редактирования транспорта

6. Поиска транспорта

**ВВЕДЕНИЕ**

Актуальность данной темы обусловлена возрастающей потребностью в автоматизации управления транспортными средствами (ТС) для эффективного планирования перевозок. В современных условиях важно иметь систему, позволяющую быстро бронировать транспорт, контролировать его доступность и корректировать данные в реальном времени.

В данной работе рассмотрены основные операции, необходимые для управления ТС: загрузка и сохранение данных в файл, бронирование ТС на определенную дату и время с учетом типа (грузовой/пассажирский) и вместимости, редактирование и отмена бронирования, поиска ТС по заданным параметрам, корректировка наличия ТС, фильтр ТС, проверка доступных ТС на заданную дату, проверка статуса ТС, удаление ТС.

Целью данной работы является создание простой и понятной в использовании программы для управления ТС, способной обрабатывать все необходимые объёмы информации.

Программа написана на языке «С», так как он является одним из самых быстрых и функциональных языков на данный момент. Кроме этого, язык, содержит стандартные библиотеки, которые предоставляют возможность не реализовывать наиболее типичные задачи, возникающие в процессе написания программы. Кроме этого, отличительной особенностью языка «С» является возможность контроля памяти, что даёт возможность программирования на системном уровне.

**1 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

**1.1 Входные и выходные данные**

**Входные данные:**

#define MAX\_STRING 50 // Максимальная длина строки

#define EXPORT\_FILE "export.txt" // Файл для экспорта

// Перечисление типов ТС

typedef enum TransportType

{

TRUCK, // Грузовой транспорт

PASSENGER // Пассажирский транспорт

};

// Структура для хранения информации о бронировании

typedef struct Booking

{

char date[20]; // Дата ЧЧ:ММ:ГГГГ

char time[10]; // Время ЧЧ:ММ

char user[MAX\_STRING]; // Имя заказчика

};

// Структура для хранения информации о транспортном средстве

typedef struct Transport

{

int id; // Уникальный идентификатор транспорта

TransportType type; // Тип транспорта

char model[MAX\_STRING]; // Модель транспорта

int capacity; // Вместимость транспорта

int available; // Количество доступных единиц ТС

char current\_status[MAX\_STRING];// Текущий статус ТС

Booking\* bookings; // Массив бронирований

int booking\_count; // Количество бронирований для ТС

};

FILE\* file; // Файл для записи

int again = 1; //Переменная для зацикливания

int choice = 0; //Переменная для выбора операции

int transport\_count = 0;//Переменная для текущего количества ТС

int transport\_capacity = 0;//Переменная для количества ТС

“Car.txt” – файл, в котором хранится начальная информация ТС.

**Выходные данные:**

Выходными данными являются два файла:

“data.txt” – файл, в который заносится временная информацию, которую вводит пользователь.

“export.txt” – файл, в который заносится информацию, которую сохранил пользователь.

**2 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

## **2.1 Разработка схем алгоритмов**

Схема алгоритма main() представлена в приложении А.

Функция deleteTransport() позволяет бронировать ТС, схема алгоритма представлена в приложении Б.

Функция loadFromFile(const char\* filename) позволяет загружать информацию из текстового файла, схема алгоритма представлена в приложении В.

**2.2 Разработка алгоритмов**

Функция orderTransport() позволяет пользователю забронировать ТС.

Алгоритм по шагам:

1) Начало

2) Объявить переменные:

char input – буфер для ввода.

int day, month, year, hour, minute – переменные для хранения даты и времени.

char user – имя заказчика.

int found – флаг, найдено ли ТС

int search\_by\_id – флаг для поиска по id

int id – переменная для хранения id

char\* endptr – указатель на символ

char time\_str – переменная для времени

3) Вывести сообщение: «Enter vehicle ID or model: ».

4) Ввод input с клавиатуры.

5) Вызвать функцию strol и передать ей параметры input и endptr.

6) Начало цикла с параметром i, от 0 до transport\_count.

7) Проверить соответствие id или model.

8) Если найдено совпадение, то установить флаг found = 1, иначе вывести сообщение: «No available vehicles of this type» и переход на шаг 28.

9) Начало цикла do – while пока !isValidDate(day, month, year).

10) Вывести сообщение: «Enter booking date (DD.MM.YYYY): »

11) Ввод day, month, year с клавиатуры.

12) Конец цикла do – while.

13) Преобразовать дату в строку.

14) Вызвать функцию isVehicleAvailable() для проверки доступности ТС на заданную дату. Если ТС забронировано, то вывести сообщение: «This vehicle is already booked on» и перейти на шаг 28.

15) Начало цикла do – while пока !isValidTime(hour, minute).

16) Вывести сообщение: «Enter booking time (HH:MM): ».

17) Ввод hour и time с клавиатуры.

18) Конец цикла do – while.

19) Преобразовать время в строку.

20) Вывести сообщение: «Enter your name:».

21) Ввод user с клавиатуры.

22) Вызвать функцию realloc и перераспределить память.

23) Скопировать дату, время и имя в новое бронирование, вызвав функцию strcpy\_s передав ей параметры data\_str, time\_str, user.

24) Изменить статус ТС на «Booked».

25) Вывести сообщение: «Successfully booked».

26) Вызвать функцию saveToFile().

27) Конец цикла по параметру i.

28) Конец.

Функция saveToFile() позволяет сохранять информацию в текстовый файл.

Алгоритм по шагам:

1) Начало.

2) Объявить переменные:

FILE\* file – указатель на текстовый файл

3) Открыть тестовый файл с именем FILENAME, вызвав функцию fopen\_s передав ей параметры file и FILENAME.

4) Начало цикла с параметром i от 0 до transport\_count.

5) Запись информации о ТС в текстовый фай.

6) Начало цикла с параметром j от 0 до booking.count.

7) Запись информации о бронировании для ТС в текстовый фал.

8) Конец цикла с параметром j.

9) Конец цикла с параметром i.

10) Вызвать функцию fclose(file).

11) Конец.

**3 РЕЗУЛЬТАТ РАБОТЫ**

Пользовательское меню для выбора действий представлена на рисунке 3.1.

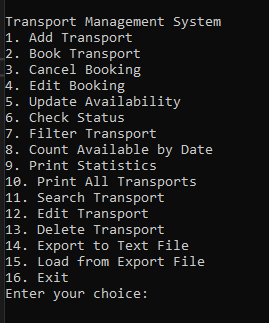


Рисунок 3.1 – Пользовательское меню

Пользовательское меню для использования функций добавления транспорта представлена на рисунках 3.2, 3.3.

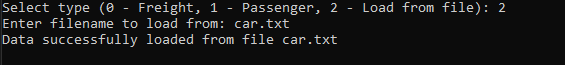


Рисунок 3.2 – Добавление ТС из файла

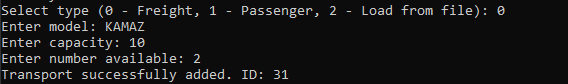


Рисунок 3.3 – Добавление ТС вручную

Пользовательское меню для использования функций бронирования ТС представлена на рисунке 3.4.

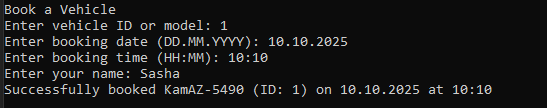


Рисунок 3.4 – Бронирование ТС

Функция вывода информации о всех ТС представлена на рисунке 3.5.

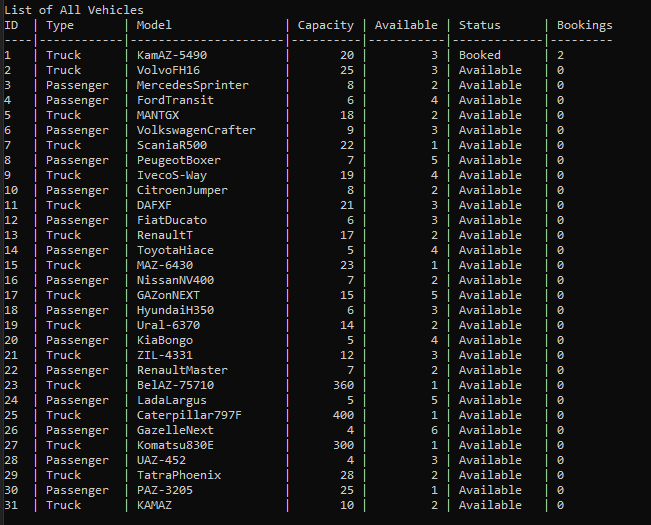


Рисунок 3.5 – Информации о всех ТС

Функция экспорта в файл представлена на рисунках 3.6 и 3.7.



Рисунок 3.6 – Экспорт в текстовый файл

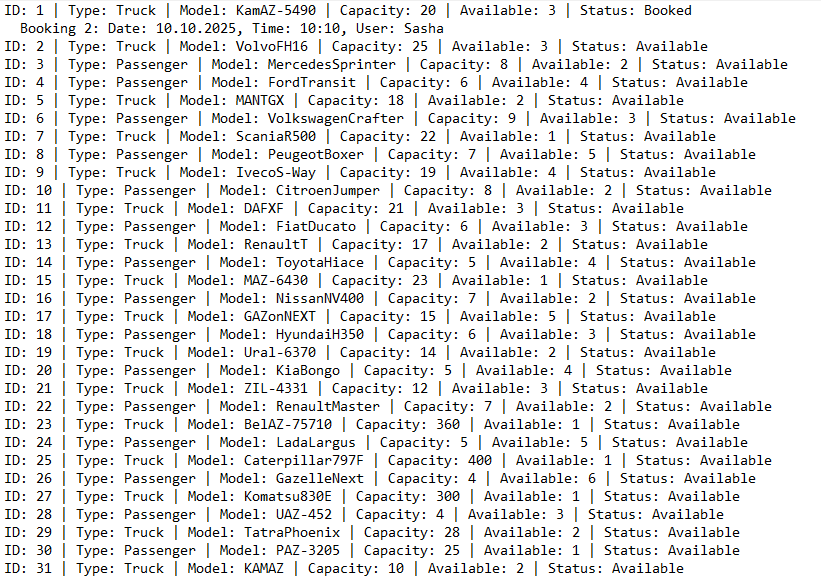


Рисунок 3.7 – Экспорт в текстовый файл

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы была успешно разработана программа для поиска маршрутов между городами. Приложение предоставляет пользователям точную информацию о времени в пути, расстоянии между населёнными пунктами и вариантах пересадок при отсутствии прямого рейса.

Программа имеет удобный и интуитивно понятный интерфейс, что значительно упрощает её использование. Чёткая структура меню, наглядное отображение маршрутов и простая система управления позволяют пользователям быстро освоить основные функции приложения.

Особое внимание при разработке было уделено оптимизации производительности. Эффективная работа с файлами данных и продуманные алгоритмы поиска обеспечивают высокую скорость работы приложения даже при обработке больших объёмов информации.

В результате выполнения проекта были не только закреплены навыки программирования на языке C, но и получен ценный опыт создания реального приложения для работы с транспортными маршрутами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Луцик, Ю. А. Основы алгоритмизации и программирования : язык Си : учебно-метод. пособие / Ю. А. Луцик, А. М. Ковальчук, Е. А. Сасин. – Минск : БГУИР, 2015. – 170с. : ил.

2. Основы алгоритмизации и программирования : лаборатор. практикум для студентов специальности 1-40 02 01 «Вычисл. машины, системы и сети» всех форм обучения. В 2 ч. Ч. 2 / сост. Ю. А. Луцик [и др.]. – Минск : БГУИР, 2010. – 36 с. : ил.

**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Схема алгоритма main()

**ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Схема алгоритма функции deleteTransport()

**ПРИЛОЖЕНИЕ В**

*(обязательное)*

Схема алгоритма функции loadFromFile(const char\* filename)

**ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Код файла main.cpp

#include "transport.h"

int main()

{

setlocale(LC\_ALL, "RUS");

int again = 1;

while (again)

{

TransportSystem();

menu();

again = againf();

cleanupTransportSystem();

}

return 0;

}

**ПРИЛОЖЕНИЕ Д**

*(обязательное)*

Код файла transport.h

#include <locale.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <ctype.h>

#define MAX\_STRING 50 // Максимальная длина строки

#define EXPORT\_FILE "export.txt" // Файл для экспорта

// Перечисление типов ТС

typedef enum TransportType

{

TRUCK, // Грузовой транспорт

PASSENGER // Пассажирский транспорт

};

// Структура для хранения информации о бронировании

typedef struct Booking

{

char date[20]; // Дата ЧЧ:ММ:ГГГГ

char time[10]; // Время ЧЧ:ММ

char user[MAX\_STRING]; // Имя заказчика

};

// Структура для хранения информации о транспортном средстве

typedef struct Transport

{

int id; // Уникальный идентификатор ТС

TransportType type; // Тип ТС

char model[MAX\_STRING]; // Модель ТС

int capacity; // Вместимость ТС

int available; // Количество доступных единиц данного ТС

char current\_status[MAX\_STRING]; // Текущий статус ТС

Booking\* bookings; // Указатель на динамический массив бронирований для данного транспорта

int booking\_count; // Количество бронирований для данного ТС

};

void exportToTextFile(); // Экспорт данных во временный текстовый файл

void saveToFile(); // Сохранение данных в файл

int loadFromFile(const char\* filename); // Загрузка данных из файла

int loadFromExportFile(); // Экспорт данных в текстовый файл

int isLeapYear(int year); // Проверка високосного года

int isValidDate(int day, int month, int year); // Корректность даты

int isValidTime(int hour, int minute); // Корректность времени

int isVehicleAvailable(int id, const char\* date); // Доступность ТС

void addTransport(); // Добавление нового ТС

void orderTransport(); // Бронирование ТС

void editBooking(); // Редактирование бронирования

void cancelBooking(); // Отмена бронирования

void updateAvailability(); // Обновление доступности ТС

void checkStatus(); // Проверка статуса ТС

void filterTransport(); // Фильтр ТС

void countAvailableByDate(); // Подсчет доступных ТС на дату

void printStatistics(); // Вывод статистики по ТС

void printTransports(); // Печать списка всех ТС

void searchTransport(); // Поиск ТС

void editTransport(); // Редактирование информации о ТС

void deleteTransport(); // Удаление ТС

void validateTransportData(); // Проверка целостности данных

void TransportSystem(); // Выделение памяти

void ensureCapacity(); // Перераспределение памяти

void cleanupTransportSystem(); // Очистка памяти

void menu(); // Меню

int againf(); // Повтор программы

**ПРИЛОЖЕНИЕ Е**

(обязательное)

Код файла function.cpp

#include "transport.h"

// Проверка високосного года

int isLeapYear(int year)

{

return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

}

// Проверка корректности даты

int isValidDate(int day, int month, int year)

{

time\_t t = time(NULL);

struct tm current\_time;

if (localtime\_s(&current\_time, &t) != 0)

{

printf("Error getting current date\n");

return 0;

}

int current\_year = current\_time.tm\_year + 1900;

int current\_month = current\_time.tm\_mon + 1;

int current\_day = current\_time.tm\_mday;

// Проверка года

if (year < current\_year || year > 2100)

{

printf("Error: Invalid year (%d). Year must be between %d and 2100.\n",

year, current\_year);

return 0;

}

// Проверка месяца

if (month < 1 || month > 12)

{

printf("Error: Invalid month (%d). Must be between 1 and 12.\n", month);

return 0;

}

// Количество дней в месяцах

int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

// Корректировка для високосного года

if (isLeapYear(year))

{

daysInMonth[1] = 29;

}

// Проверка дня

if (day < 1 || day > daysInMonth[month - 1])

{

printf("Error: Invalid day (%d) for month %d.\n", day, month);

return 0;

}

// Проверка, что дата не в прошлом

if (year == current\_year)

{

if (month < current\_month || (month == current\_month && day < current\_day))

{

printf("Error: Date %02d.%02d.%04d cannot be earlier than the current date.\n",

day, month, year);

return 0;

}

}

return 1;

}

// Проверка корректности времени

int isValidTime(int hour, int minute)

{

if (hour < 0 || hour > 23)

{

printf("Error: Invalid hour (%d). Must be between 0 and 23.\n", hour);

return 0;

}

if (minute < 0 || minute > 59)

{

printf("Error: Invalid minute (%d). Must be between 0 and 59.\n", minute);

return 0;

}

return 1;

}

//Меню

void menu()

{

int choice;

do {

printf("\nTransport Management System\n");

printf("1. Add Transport\n");

printf("2. Book Transport\n");

printf("3. Cancel Booking\n");

printf("4. Edit Booking\n");

printf("5. Update Availability\n");

printf("6. Check Status\n");

printf("7. Filter Transport\n");

printf("8. Count Available by Date\n");

printf("9. Print Statistics\n");

printf("10. Print All Transports\n");

printf("11. Search Transport\n");

printf("12. Edit Transport\n");

printf("13. Delete Transport\n");

printf("14. Export to Text File\n");

printf("15. Load from Export File\n");

printf("16. Exit\n");

printf("Enter your choice: ");

if (scanf\_s("%d", &choice) != 1)

{

while (getchar() != '\n');

printf("Invalid input\n");

continue;

}

while (getchar() != '\n');

switch (choice)

{

case 1: addTransport(); break;

case 2: orderTransport(); break;

case 3: cancelBooking(); break;

case 4: editBooking(); break;

case 5: updateAvailability(); break;

case 6: checkStatus(); break;

case 7: filterTransport(); break;

case 8: countAvailableByDate(); break;

case 9: printStatistics(); break;

case 10: printTransports(); break;

case 11: searchTransport(); break;

case 12: editTransport(); break;

case 13: deleteTransport(); break;

case 14: exportToTextFile(); break;

case 15:

if (loadFromExportFile())

{

printf("Data loaded from export file\n");

}

break;

case 16:

printf("Exiting...\n");

break;

default:

printf("Invalid choice\n");

}

} while (choice != 16);

}

// Повтор программы

int againf()

{

int again, x;

printf("Enter 1 to continue or any other key to exit: ");

while (1)

{

x = scanf\_s("%d", &again);

if (x != 1 || again != 1)

{

printf("\n");

printf("Program is terminating.\n");

again = 0;

return again;

}

else

{

printf("Restarting the program\n");

break;

}

}

return again;

}

**ПРИЛОЖЕНИЕ Ж**

(обязательное)

Код файла transport.cpp

#include "transport.h"

const char\* FILENAME = (char\*)"data.txt"; // Указание на именя файла для хранения данных

Transport\* transports = NULL; // Указатель на массив структур

int transport\_count = 0; // Фактическое количество ТС

int transport\_capacity = 0; // Вместимость массива

// Инициализация транспортной системы

void TransportSystem()

{

transport\_capacity = 100; // Начальная вместительность массива

transports = (Transport\*)malloc(transport\_capacity \* sizeof(Transport));

for (int i = 0; i < transport\_capacity; i++)

{

(\*(transports + i)).bookings = NULL;

(\*(transports + i)).booking\_count = 0;

}

}

// Увеличение памяти

void ensureCapacity()

{

if (transport\_count >= transport\_capacity)

{

transport\_capacity \*= 2;

Transport\* new\_transports = (Transport\*)realloc(transports, transport\_capacity \* sizeof(Transport));

if (new\_transports)

{

transports = new\_transports;

}

}

}

// Очистка памяти

void cleanupTransportSystem()

{

if (transports)

{

for (int i = 0; i < transport\_count; i++)

{

if ((\*(transports + i)).bookings)

{

free((\*(transports + i)).bookings);

}

}

free(transports);

transports = NULL;

}

transport\_count = 0;

transport\_capacity = 0;

}

// Проверка корректности данных о транспортах

void validateTransportData()

{

for (int i = 0; i < transport\_count; i++)

{

// Проверка id

if ((\*(transports + i)).id <= 0)

{

(\*(transports + i)).id = i + 1;

}

// Проверка типа

if ((\*(transports + i)).type != TRUCK && (\*(transports + i)).type != PASSENGER)

{

(\*(transports + i)).type = TRUCK;

}

// Проверка модели

if (strlen((\*(transports + i)).model) == 0)

{

strcpy\_s((\*(transports + i)).model, sizeof((\*(transports + i)).model), "Unknown");

}

// Проверка вместительности

if ((\*(transports + i)).capacity < 0)

{

(\*(transports + i)).capacity = 0;

}

// Проверка количества

if ((\*(transports + i)).available < 0)

{

(\*(transports + i)).available = 0;

}

// Проверка статуса

if (strlen((\*(transports + i)).current\_status) == 0)

{

strcpy\_s((\*(transports + i)).current\_status, sizeof((\*(transports + i)).current\_status), "Available");

}

}

}

// Доступность

int isVehicleAvailable(int id, const char\* date)

{

for (int i = 0; i < transport\_count; i++)

{

if ((\*(transports + i)).id == id)

{

for (int j = 0; j < (\*(transports + i)).booking\_count; j++)

{

if (strcmp((\*(transports + i)).bookings[j].date, date) == 0)

{

return 0;

}

}

return 1;

}

}

return 0;

}

// Бронирование ТС

void orderTransport()

{

char input[MAX\_STRING]; // Буфер для ввода

int day, month, year, hour, minute; // Переменные для хранения даты и времени

char user[MAX\_STRING]; // Имя заказчика

int found = 0; // Флаг, найден транспорт

int search\_by\_id = 0; // Флаг, поиска по ID

int id = 0; // Переменная для хранения ID

printf("\nBook a Vehicle\nEnter vehicle ID or model: ");

if (fgets(input, sizeof(input), stdin) == NULL)

{

return;

}

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

for (int i = 0; i < transport\_count; i++)

{

// Поиск по ID или модели

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strcmp((\*(transports + i)).model, input) == 0))

{

found = 1;

if ((\*(transports + i)).available <= 0)

{

printf("No available vehicles of this type\n");

return;

}

do

{

printf("Enter booking date (DD.MM.YYYY): ");

if (scanf\_s("%d.%d.%d", &day, &month, &year) != 3)

{

while (getchar() != '\n');

printf("Invalid date format\n");

continue;

}

while (getchar() != '\n');

} while (!isValidDate(day, month, year));

char date\_str[20];

sprintf\_s(date\_str, sizeof(date\_str), "%02d.%02d.%04d", day, month, year);

// Проверка доступности ТС на заданную дату

if (!isVehicleAvailable((\*(transports + i)).id, date\_str))

{

printf("This vehicle is already booked on %s\n", date\_str);

return;

}

do

{

printf("Enter booking time (HH:MM): ");

if (scanf\_s("%d:%d", &hour, &minute) != 2)

{

while (getchar() != '\n');

printf("Invalid time format\n");

continue;

}

while (getchar() != '\n');

} while (!isValidTime(hour, minute));

char time\_str[10];

sprintf\_s(time\_str, sizeof(time\_str), "%02d:%02d", hour, minute);

printf("Enter your name: ");

if (fgets(user, sizeof(user), stdin) == NULL)

{

return;

}

user[strcspn(user, "\n")] = '\0';

(\*(transports + i)).booking\_count++;

Booking\* new\_bookings = (Booking\*)realloc((\*(transports + i)).bookings,

(\*(transports + i)).booking\_count \* sizeof(Booking));

if (new\_bookings)

{

(\*(transports + i)).bookings = new\_bookings;

}

else

{

printf("Memory allocation failed\n");

return;

}

// Копирование даты, времени и имя пользователя в новое бронирование

strcpy\_s((\*(transports + i)).bookings[(\*(transports + i)).booking\_count - 1].date,

sizeof((\*(transports + i)).bookings[0].date), date\_str);

strcpy\_s((\*(transports + i)).bookings[(\*(transports + i)).booking\_count - 1].time,

sizeof((\*(transports + i)).bookings[0].time), time\_str);

strcpy\_s((\*(transports + i)).bookings[(\*(transports + i)).booking\_count - 1].user,

sizeof((\*(transports + i)).bookings[0].user), user);

(\*(transports + i)).available--;

strcpy\_s((\*(transports + i)).current\_status,

sizeof((\*(transports + i)).current\_status), "Booked");

printf("Successfully booked %s (ID: %d) on %s at %s\n",

(\*(transports + i)).model, (\*(transports + i)).id, date\_str, time\_str);

saveToFile();

return;

}

}

if (!found) printf("Vehicle not found!\n");

}

// Отмена бронирования

void cancelBooking()

{

char input[MAX\_STRING]; // Буфер для ввода

int found = 0; // Флаг, найден ли транспорт

int search\_by\_id = 0; // Флаг, для поиска

int id = 0; // Переменная для хранения ID

int booking\_num; // Переменная для номера бронирования

printf("\nCancel Booking\nEnter vehicle ID or model: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

for (int i = 0; i < transport\_count; i++)

{

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strcmp((\*(transports + i)).model, input) == 0))

{

found = 1;

if ((\*(transports + i)).booking\_count == 0)

{

printf("No bookings found for this vehicle\n");

return;

}

printf("\nCurrent bookings for %s (ID: %d):\n", (\*(transports + i)).model, (\*(transports + i)).id);

for (int j = 0; j < (\*(transports + i)).booking\_count; j++)

{

printf("%d. Date: %s, Time: %s, User: %s\n",

j + 1, (\*(transports + i)).bookings[j].date,

(\*(transports + i)).bookings[j].time, (\*(transports + i)).bookings[j].user);

}

printf("Enter booking number to cancel: ");

scanf\_s("%d", &booking\_num);

while (getchar() != '\n');

if (booking\_num < 1 || booking\_num >(\*(transports + i)).booking\_count)

{

printf("Invalid booking number\n");

return;

}

for (int j = booking\_num - 1; j < (\*(transports + i)).booking\_count - 1; j++)

{

(\*(transports + i)).bookings[j] = (\*(transports + i)).bookings[j + 1];

}

(\*(transports + i)).booking\_count--;

(\*(transports + i)).available++;

if ((\*(transports + i)).booking\_count == 0)

{

strcpy\_s((\*(transports + i)).current\_status, "Available");

}

printf("Booking cancelled successfully\n");

saveToFile();

return;

}

}

if (!found) printf("Vehicle not found!\n");

}

// Редактирование бронирования

void editBooking()

{

char input[MAX\_STRING]; // Буфер для ввода

int found = 0; // Флаг, найден ли транспорт

int search\_by\_id = 0; // Флаг, поиска по ID или по модели

int id = 0; // Переменная для хранения ID

int booking\_num; // Переменная для номера бронирования

int day, month, year, hour, minute; // Переменные для хранения даты и времени

char date\_input[20]; // Буфер для даты

char new\_date[20]; // Буфер для новой даты

char time\_input[10]; // Буфер для времени

char new\_time[10]; // Буфер для нового времени

printf("\nEdit Booking\nEnter vehicle ID or model: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

for (int i = 0; i < transport\_count; i++)

{

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strcmp((\*(transports + i)).model, input) == 0))

{

found = 1;

if ((\*(transports + i)).booking\_count == 0)

{

printf("No bookings found for this vehicle\n");

return;

}

printf("\nCurrent bookings for %s (ID: %d):\n", (\*(transports + i)).model, (\*(transports + i)).id);

for (int j = 0; j < (\*(transports + i)).booking\_count; j++)

{

printf("%d. Date: %s, Time: %s, User: %s\n",

j + 1, (\*(transports + i)).bookings[j].date,

(\*(transports + i)).bookings[j].time, (\*(transports + i)).bookings[j].user);

}

printf("Enter booking number to edit: ");

scanf\_s("%d", &booking\_num);

while (getchar() != '\n');

if (booking\_num < 1 || booking\_num >(\*(transports + i)).booking\_count)

{

printf("Invalid booking number\n");

return;

}

Booking\* booking = &(\*(transports + i)).bookings[booking\_num - 1];

// Изменение даты бронирования

do

{

printf("Enter new booking date (DD.MM.YYYY) [current: %s]: ", booking->date);

fgets(date\_input, sizeof(date\_input), stdin);

date\_input[strcspn(date\_input, "\n")] = '\0';

if (strlen(date\_input) == 0)

break;

FILE\* fdate = NULL;

if (tmpfile\_s(&fdate) != 0 || fdate == NULL)

{

printf("Temporary file error\n");

return;

}

fputs(date\_input, fdate);

rewind(fdate);

if (fscanf\_s(fdate, "%d.%d.%d", &day, &month, &year) != 3)

{

printf("Invalid date format\n");

fclose(fdate);

continue;

}

fclose(fdate);

if (!isValidDate(day, month, year))

{

printf("Invalid date\n");

continue;

}

sprintf\_s(new\_date, sizeof(new\_date), "%02d.%02d.%04d", day, month, year);

int conflict = 0;

for (int j = 0; j < (\*(transports + i)).booking\_count; j++)

{

if (j != booking\_num - 1 && strcmp((\*(transports + i)).bookings[j].date, new\_date) == 0)

{

conflict = 1;

break;

}

}

if (conflict)

{

printf("Vehicle already booked on this date\n");

continue;

}

strcpy\_s(booking->date, sizeof(booking->date), new\_date);

break;

} while (1);

// Изменение времени бронирования

do

{

printf("Enter new booking time (HH:MM) [current: %s]: ", booking->time);

fgets(time\_input, sizeof(time\_input), stdin);

time\_input[strcspn(time\_input, "\n")] = '\0';

if (strlen(time\_input) == 0)

break;

FILE\* ftime = NULL;

if (tmpfile\_s(&ftime) != 0 || ftime == NULL)

{

printf("Temporary file error\n");

return;

}

fputs(time\_input, ftime);

rewind(ftime);

if (fscanf\_s(ftime, "%d:%d", &hour, &minute) != 2)

{

printf("Invalid time format\n");

fclose(ftime);

continue;

}

fclose(ftime);

if (!isValidTime(hour, minute))

{

printf("Invalid time\n");

continue;

}

sprintf\_s(new\_time, sizeof(new\_time), "%02d:%02d", hour, minute);

strcpy\_s(booking->time, sizeof(booking->time), new\_time);

break;

} while (1);

printf("Enter new user name [current: %s]: ", booking->user);

char user\_input[MAX\_STRING];

fgets(user\_input, sizeof(user\_input), stdin);

user\_input[strcspn(user\_input, "\n")] = '\0';

if (strlen(user\_input) > 0)

{

strcpy\_s(booking->user, sizeof(booking->user), user\_input);

}

printf("Booking updated successfully\n");

saveToFile();

return;

}

}

if (!found)

printf("Vehicle not found!\n");

}

// Сохранение данных в файл

void saveToFile()

{

FILE\* file;

if (fopen\_s(&file, FILENAME, "w") != 0 || file == NULL)

{

printf("Error opening file for writing\n");

return;

}

for (int i = 0; i < transport\_count; i++)

{

fprintf(file, "%d %d %s %d %d %s %d\n",

(\*(transports + i)).id,

(\*(transports + i)).type,

(\*(transports + i)).model,

(\*(transports + i)).capacity,

(\*(transports + i)).available,

(\*(transports + i)).current\_status,

(\*(transports + i)).booking\_count);

for (int j = 0; j < (\*(transports + i)).booking\_count; j++)

{

fprintf(file, "%s %s %s\n",

(\*(transports + i)).bookings[j].date,

(\*(transports + i)).bookings[j].time,

(\*(transports + i)).bookings[j].user);

}

}

fclose(file);

}

// Загрузка данных из файла

int loadFromFile(const char\* filename)

{

FILE\* file;

if (fopen\_s(&file, filename, "r") != 0 || file == NULL)

{

printf("Error opening file %s\n", filename);

return 0;

}

while (!feof(file))

{

Transport t = { 0 };

int booking\_count = 0; // Количество бронирований

if (fscanf\_s(file, "%d %d %49s %d %d %49s %d\n",

&t.id,

(int\*)&t.type,

t.model, (unsigned)\_countof(t.model),

&t.capacity,

&t.available,

t.current\_status, (unsigned)\_countof(t.current\_status),

&booking\_count) != 7)

{

break;

}

t.booking\_count = booking\_count;

t.bookings = (Booking\*)malloc(booking\_count \* sizeof(Booking));

for (int i = 0; i < booking\_count; i++)

{

if (fscanf\_s(file, "%19s %9s %49[^\n]\n",

t.bookings[i].date, (unsigned)\_countof(t.bookings[i].date),

t.bookings[i].time, (unsigned)\_countof(t.bookings[i].time),

t.bookings[i].user, (unsigned)\_countof(t.bookings[i].user)) != 3)

{

break;

}

}

ensureCapacity();

transports[transport\_count++] = t;

}

fclose(file);

validateTransportData();

return 1;

}

// Экспорт данных в текстовый файл

int loadFromExportFile()

{

FILE\* file;

if (fopen\_s(&file, EXPORT\_FILE, "r") != 0 || file == NULL)

{

printf("Error opening export file\n");

return 0;

}

cleanupTransportSystem();

TransportSystem();

char line[256]; // Буфер для строк из файла

while (fgets(line, sizeof(line), file))

{

if (strstr(line, "ID:"))

{

Transport t = { 0 };

char type\_str[20] = ""; // Строка для типа транспорта

FILE\* temp = NULL;

if (tmpfile\_s(&temp) != 0 || temp == NULL)

{

printf("Failed to create temporary file\n");

fclose(file);

return 0;

}

fputs(line, temp);

rewind(temp);

if (fscanf\_s(temp, "ID: %d | Type: %19[^|] | Model: %49[^|] | Capacity: %d | Available: %d | Status: %49[^\n]",

&t.id,

type\_str, (unsigned)\_countof(type\_str),

t.model, (unsigned)\_countof(t.model),

&t.capacity,

&t.available,

t.current\_status, (unsigned)\_countof(t.current\_status)) != 6)

{

fclose(temp);

continue;

}

fclose(temp);

if (strstr(type\_str, "Truck"))

{

t.type = TRUCK;

}

else if (strstr(type\_str, "Passenger"))

{

t.type = PASSENGER;

}

t.booking\_count = 0;

t.bookings = NULL;

while (fgets(line, sizeof(line), file))

{

if (strstr(line, "Booking"))

{

Booking b = { 0 };

FILE\* btemp = NULL;

if (tmpfile\_s(&btemp) != 0 || btemp == NULL)

{

printf("Failed to create temporary file for booking\n");

break;

}

fputs(line, btemp);

rewind(btemp);

if (fscanf\_s(btemp, " Booking %\*d: Date: %19[^,], Time: %9[^,], User: %49[^\n]",

b.date, (unsigned)\_countof(b.date),

b.time, (unsigned)\_countof(b.time),

b.user, (unsigned)\_countof(b.user)) == 3)

{

t.booking\_count++;

Booking\* new\_bookings = (Booking\*)realloc(t.bookings, t.booking\_count \* sizeof(Booking));

if (new\_bookings)

{

t.bookings = new\_bookings;

t.bookings[t.booking\_count - 1] = b;

}

else

{

printf("Memory allocation failed for bookings\n");

fclose(btemp);

break;

}

}

fclose(btemp);

}

else

{

break;

}

}

ensureCapacity();

transports[transport\_count++] = t;

}

}

fclose(file);

validateTransportData();

return 1;

}

// Экспорт данных во временный текстовый файл

void exportToTextFile()

{

FILE\* file;

if (fopen\_s(&file, EXPORT\_FILE, "w") != 0 || file == NULL)

{

printf("Error opening export file\n");

return;

}

for (int i = 0; i < transport\_count; i++)

{

fprintf(file, "ID: %d | Type: %s | Model: %s | Capacity: %d | Available: %d | Status: %s\n",

(\*(transports + i)).id,

(\*(transports + i)).type == TRUCK ? "Truck" : "Passenger",

(\*(transports + i)).model,

(\*(transports + i)).capacity,

(\*(transports + i)).available,

(\*(transports + i)).current\_status);

for (int j = 0; j < (\*(transports + i)).booking\_count; j++)

{

fprintf(file, " Booking %d: Date: %s, Time: %s, User: %s\n",j + 1,

(\*(transports + i)).bookings[j].date,

(\*(transports + i)).bookings[j].time,

(\*(transports + i)).bookings[j].user);

}

}

fclose(file);

printf("Data exported to %s\n", EXPORT\_FILE);

}

// Добавление нового ТС

void addTransport()

{

ensureCapacity();

if (transport\_count >= transport\_capacity)

{

printf("Maximum number of transports reached\n");

return;

}

int type\_input; // Переменная для выбора типа ввода

while (1)

{

printf("Select type (0 - Freight, 1 - Passenger, 2 - Load from file): ");

if (scanf\_s("%d", &type\_input) != 1)

{

printf("Input error! Please enter 0, 1, or 2.\n");

while (getchar() != '\n');

continue;

}

if (type\_input == 0 || type\_input == 1)

{

Transport t = { 0 };

t.id = transport\_count > 0 ? transports[transport\_count - 1].id + 1 : 1;

t.type = (TransportType)type\_input;

t.booking\_count = 0;

t.bookings = NULL;

// Ввод модели

while (1)

{

printf("Enter model: ");

char model\_input[MAX\_STRING]; // Буфер для ввода модели

while (getchar() != '\n');

if (fgets(model\_input, sizeof(model\_input), stdin) == NULL)

{

printf("Input error!\n");

continue;

}

model\_input[strcspn(model\_input, "\n")] = '\0';

if (strlen(model\_input) == 0)

{

printf("Model cannot be empty\n");

continue;

}

strcpy\_s(t.model, sizeof(t.model), model\_input);

break;

}

// Ввод вместительности

while (1)

{

printf("Enter capacity: ");

if (scanf\_s("%d", &t.capacity) != 1)

{

printf("Capacity input error. Please enter an integer.\n");

while (getchar() != '\n');

continue;

}

if (t.capacity < 0)

{

printf("Capacity cannot be negative\n");

continue;

}

break;

}

// Ввод количества ТС

while (1)

{

printf("Enter number available: ");

if (scanf\_s("%d", &t.available) != 1)

{

printf("Availability input error. Please enter an integer.\n");

while (getchar() != '\n');

continue;

}

if (t.available < 1)

{

printf("Quantity must be greater than 0\n");

continue;

}

break;

}

strcpy\_s(t.current\_status, sizeof(t.current\_status), "Available");

transports[transport\_count++] = t;

saveToFile();

printf("Transport successfully added. ID: %d\n", t.id);

return;

}

// Ввод с файла

else if (type\_input == 2)

{

while (getchar() != '\n');

char filename[MAX\_STRING]; // Буфер для имени файла

printf("Enter filename to load from: ");

if (fgets(filename, sizeof(filename), stdin) == NULL)

{

printf("Filename input error\n");

continue;

}

filename[strcspn(filename, "\n")] = '\0';

if (strlen(filename) == 0)

{

strcpy\_s(filename, sizeof(filename), "car.txt");

}

if (loadFromFile(filename))

{

printf("Data successfully loaded from file %s\n", filename);

return;

}

else

{

printf("Error loading from file %s. Please try again.\n", filename);

continue;

}

}

else

{

printf("Invalid type! Please enter 0, 1, or 2.\n");

}

}

}

// Обновление доступности ТС

void updateAvailability()

{

char input[MAX\_STRING]; // Буфер для ввода

int found = 0; // Флаг, найден ли транспорт

int search\_by\_id = 0; // Флаг: поиск по ID или по модели

int id = 0; // Переменная для хранения ID

int new\_available; // Новое количество доступных ТС

printf("\nUpdate Availability\nEnter vehicle ID or model: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

for (int i = 0; i < transport\_count; i++)

{

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strcmp((\*(transports + i)).model, input) == 0))

{

found = 1;

printf("Current availability: %d\n", (\*(transports + i)).available);

printf("Enter new available quantity: ");

if (scanf\_s("%d", &new\_available) != 1)

{

printf("Input error! Please enter an integer.\n");

while (getchar() != '\n');

return;

}

while (getchar() != '\n');

if (new\_available < 0)

{

printf("Quantity cannot be negative\n");

return;

}

(\*(transports + i)).available = new\_available;

if (new\_available == 0)

{

strcpy\_s((\*(transports + i)).current\_status, sizeof((\*(transports + i)).current\_status), "Unavailable");

}

else if ((\*(transports + i)).booking\_count > 0)

{

strcpy\_s((\*(transports + i)).current\_status, sizeof((\*(transports + i)).current\_status), "Booked");

}

else

{

strcpy\_s((\*(transports + i)).current\_status, sizeof((\*(transports + i)).current\_status), "Available");

}

printf("Availability updated successfully\n");

saveToFile();

return;

}

}

if (!found) printf("Vehicle not found!\n");

}

// Проверка статуса ТС

void checkStatus()

{

char input[MAX\_STRING]; // Буфер для ввода

int found = 0; // Флаг, найден ли транспорт

int search\_by\_id = 0; // Флаг поиска по ID или модели

int id = 0; // Переменная для ID

printf("\nCheck Status\nEnter vehicle ID or model: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

for (int i = 0; i < transport\_count; i++)

{

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strcmp((\*(transports + i)).model, input) == 0))

{

found = 1;

printf("\nVehicle Information:\n");

printf("ID: %d\n", (\*(transports + i)).id);

printf("Type: %s\n", (\*(transports + i)).type == TRUCK ? "Truck" : "Passenger");

printf("Model: %s\n", (\*(transports + i)).model);

printf("Capacity: %d\n", (\*(transports + i)).capacity);

printf("Available: %d\n", (\*(transports + i)).available);

printf("Status: %s\n", (\*(transports + i)).current\_status);

if ((\*(transports + i)).booking\_count > 0)

{

printf("\nBookings:\n");

for (int j = 0; j < (\*(transports + i)).booking\_count; j++)

{

printf("%d. Date: %s, Time: %s, User: %s\n",

j + 1, (\*(transports + i)).bookings[j].date,

(\*(transports + i)).bookings[j].time, (\*(transports + i)).bookings[j].user);

}

}

return;

}

}

if (!found) printf("Vehicle not found!\n");

}

// Фильтр ТС

void filterTransport()

{

int type\_filter = -1; // Фильтр по типу

char model\_filter[MAX\_STRING] = ""; // Фильтр по модели

int min\_capacity = -1; // Минимальная вместимость

char type\_input[10]; // Буфер для типа

char capacity\_input[10]; // Буфер для минимальной вместимости

int found = 0; // Счётчик найденных транспортов

printf("\nFilter Vehicles\n");

printf("Enter type (0 - Truck, 1 - Passenger, Enter - any): ");

fgets(type\_input, sizeof(type\_input), stdin);

type\_input[strcspn(type\_input, "\n")] = '\0';

if (strlen(type\_input) > 0)

{

type\_filter = atoi(type\_input);

if (type\_filter < 0 || type\_filter > 1)

{

printf("Invalid type, using any type\n");

type\_filter = -1;

}

}

printf("Enter model to filter (Enter - any): ");

fgets(model\_filter, sizeof(model\_filter), stdin);

model\_filter[strcspn(model\_filter, "\n")] = '\0';

printf("Enter minimum capacity (Enter - any): ");

fgets(capacity\_input, sizeof(capacity\_input), stdin);

capacity\_input[strcspn(capacity\_input, "\n")] = '\0';

if (strlen(capacity\_input) > 0)

{

min\_capacity = atoi(capacity\_input);

if (min\_capacity < 0)

{

printf("Invalid capacity, using any capacity\n");

min\_capacity = -1;

}

}

printf("\nFilter Results:\n");

printf("ID | Type | Model | Capacity | Available | Status\n");

printf("----|------------|----------------------|----------|-----------|--------\n");

for (int i = 0; i < transport\_count; i++)

{

if ((type\_filter == -1 || (\*(transports + i)).type == type\_filter) &&

(strlen(model\_filter) == 0 || strstr((\*(transports + i)).model, model\_filter)) &&

(min\_capacity == -1 || (\*(transports + i)).capacity >= min\_capacity))

{

printf("%-3d | %-10s | %-20s | %8d | %9d | %s\n",

(\*(transports + i)).id,

(\*(transports + i)).type == TRUCK ? "Truck" : "Passenger",

(\*(transports + i)).model,

(\*(transports + i)).capacity,

(\*(transports + i)).available,

(\*(transports + i)).current\_status);

found++;

}

}

if (!found)

{

printf("No vehicles match the filter criteria\n");

}

}

// Подсчет доступных ТС на дату

void countAvailableByDate()

{

int day, month, year; // Переменные для дня, месяца и года

char date\_str[20]; // Буфер для даты

int available\_count = 0; // Счётчик доступных транспортов

printf("\nCount Available Vehicles by Date\n");

do {

printf("Enter date (DD.MM.YYYY): ");

if (scanf\_s("%d.%d.%d", &day, &month, &year) != 3)

{

while (getchar() != '\n');

printf("Invalid date format\n");

continue;

}

while (getchar() != '\n');

} while (!isValidDate(day, month, year));

printf(date\_str, "%02d.%02d.%04d", day, month, year);

for (int i = 0; i < transport\_count; i++)

{

if (isVehicleAvailable((\*(transports + i)).id, date\_str) && (\*(transports + i)).available > 0)

{

available\_count++;

}

}

printf("\nAvailable vehicles on %s: %d\n", date\_str, available\_count);

if (available\_count > 0)

{

printf("\nAvailable Vehicles:\n");

printf("ID | Type | Model | Capacity\n");

printf("----|------------|----------------------|----------\n");

for (int i = 0; i < transport\_count; i++)

{

if (isVehicleAvailable((\*(transports + i)).id, date\_str) && (\*(transports + i)).available > 0)

{

printf("%-3d | %-10s | %-20s | %8d\n",

(\*(transports + i)).id,

(\*(transports + i)).type == TRUCK ? "Truck" : "Passenger",

(\*(transports + i)).model,

(\*(transports + i)).capacity);

}

}

}

}

// Вывод статистики по ТС

void printStatistics()

{

int total\_vehicles = transport\_count; // Общее количество транспортных средств

int total\_trucks = 0; // Счётчик грузовых ТС

int total\_passengers = 0; // Счётчик пассажирских ТС

int total\_available = 0; // Общее количество доступных ТС

int total\_booked = 0; // Общее количество бронирований

int total\_capacity = 0; // Общая вместимость

for (int i = 0; i < transport\_count; i++)

{

if ((\*(transports + i)).type == TRUCK)

{

total\_trucks++;

}

else {

total\_passengers++;

}

total\_available += (\*(transports + i)).available;

total\_booked += (\*(transports + i)).booking\_count;

total\_capacity += (\*(transports + i)).capacity \* (\*(transports + i)).available;

}

printf("\nTransport Statistics\n");

printf("Total vehicles: %d\n", total\_vehicles);

printf(" - Trucks: %d\n", total\_trucks);

printf(" - Passenger vehicles: %d\n", total\_passengers);

printf("Total available vehicles: %d\n", total\_available);

printf("Total bookings: %d\n", total\_booked);

printf("Total transport capacity: %d\n", total\_capacity);

if (total\_vehicles > 0)

{

printf("Average capacity per vehicle: %.2f\n", (float)total\_capacity / total\_vehicles);

}

}

void printTransports()

{

printf("\nList of All Vehicles\n");

printf("ID | Type | Model | Capacity | Available | Status | Bookings\n");

printf("----|------------|----------------------|----------|-----------|-------------|---------\n");

for (int i = 0; i < transport\_count; i++)

{

printf("%-3d | %-10s | %-20s | %8d | %9d | %-11s | %d\n",

(\*(transports + i)).id,

(\*(transports + i)).type == TRUCK ? "Truck" : "Passenger",

(\*(transports + i)).model,

(\*(transports + i)).capacity,

(\*(transports + i)).available,

(\*(transports + i)).current\_status,

(\*(transports + i)).booking\_count);

}

}

// Поиск ТС

void searchTransport()

{

char input[MAX\_STRING]; // Буфер для ввода

int found = 0; // Флаг найденных бронирований

int search\_by\_id = 0; // Флаг поиска по ID

int id = 0; // Переменная для хранения ID

printf("\nSearch Vehicle\nEnter ID or model: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

printf("\nSearch Results:\n");

printf("ID | Type | Model | Capacity | Available | Status\n");

printf("----|------------|----------------------|----------|-----------|--------\n");

for (int i = 0; i < transport\_count; i++)

{

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strstr((\*(transports + i)).model, input)))

{

printf("%-3d | %-10s | %-20s | %8d | %9d | %s\n",

(\*(transports + i)).id,

(\*(transports + i)).type == TRUCK ? "Truck" : "Passenger",

(\*(transports + i)).model,

(\*(transports + i)).capacity,

(\*(transports + i)).available,

(\*(transports + i)).current\_status);

found++;

}

}

if (!found)

{

printf("No vehicles found matching \"%s\"\n", input);

}

}

// Редактирование информации о ТС

void editTransport()

{

char input[MAX\_STRING]; // Буфер для ввода

int found = 0; // Флаг найденного транспорта

int search\_by\_id = 0; // Флаг поиска по ID

int id = 0; // Переменная для ID

int field; // Переменная для выбора

int new\_id; // Переменная для нового Id

int new\_type; // Переменная для нового типа

char new\_model[MAX\_STRING]; // Для хранения новой модели

int new\_capacity; // Переменная для новой вместительности

int new\_available; // Переменная для новой доступности

char new\_status[MAX\_STRING]; // Для нового статуса

printf("\nEdit Vehicle\nEnter ID or model: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

for (int i = 0; i < transport\_count; i++)

{

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strcmp((\*(transports + i)).model, input) == 0))

{

found = 1;

printf("\nCurrent Vehicle Information:\n");

printf("1. ID: %d\n", (\*(transports + i)).id);

printf("2. Type: %s\n", (\*(transports + i)).type == TRUCK ? "Truck" : "Passenger");

printf("3. Model: %s\n", (\*(transports + i)).model);

printf("4. Capacity: %d\n", (\*(transports + i)).capacity);

printf("5. Available: %d\n", (\*(transports + i)).available);

printf("6. Status: %s\n", (\*(transports + i)).current\_status);

printf("\nEnter field number to edit (1-6, 0 to cancel): ");

scanf\_s("%d", &field);

while (getchar() != '\n');

if (field == 0)

{

printf("Edit cancelled\n");

return;

}

switch (field)

{

case 1:

{

printf("Enter new ID: ");

scanf\_s("%d", &new\_id);

while (getchar() != '\n');

for (int j = 0; j < transport\_count; j++)

{

if (j != i && transports[j].id == new\_id)

{

printf("ID %d already exists\n", new\_id);

return;

}

}

(\*(transports + i)).id = new\_id;

break;

}

case 2:

{

printf("Enter new type (0 - Truck, 1 - Passenger): ");

scanf\_s("%d", &new\_type);

while (getchar() != '\n');

if (new\_type == 0 || new\_type == 1)

{

(\*(transports + i)).type = (TransportType)new\_type;

}

else

{

printf("Invalid type\n");

return;

}

break;

}

case 3:

{

printf("Enter new model: ");

fgets(new\_model, sizeof(new\_model), stdin);

new\_model[strcspn(new\_model, "\n")] = '\0';

if (strlen(new\_model) > 0)

{

strcpy\_s((\*(transports + i)).model, new\_model);

}

else {

printf("Model cannot be empty\n");

return;

}

break;

}

case 4:

{

printf("Enter new capacity: ");

scanf\_s("%d", &new\_capacity);

while (getchar() != '\n');

if (new\_capacity > 0)

{

(\*(transports + i)).capacity = new\_capacity;

}

else {

printf("Capacity must be positive\n");

return;

}

break;

}

case 5:

{

printf("Enter new available count: ");

scanf\_s("%d", &new\_available);

while (getchar() != '\n');

if (new\_available >= 0 && new\_available <= (\*(transports + i)).capacity)

{

(\*(transports + i)).available = new\_available;

}

else {

printf("Available must be between 0 and capacity\n");

return;

}

break;

}

case 6:

{

printf("Enter new status: ");

fgets(new\_status, sizeof(new\_status), stdin);

new\_status[strcspn(new\_status, "\n")] = '\0';

if (strlen(new\_status) > 0)

{

strcpy\_s((\*(transports + i)).current\_status, new\_status);

}

else {

printf("Status cannot be empty\n");

return;

}

break;

}

default:

printf("Invalid field number\n");

return;

}

printf("Vehicle updated successfully\n");

return;

}

}

if (!found)

{

printf("Vehicle not found\n");

}

}

// Удаление ТС

void deleteTransport()

{

char input[MAX\_STRING]; // Буфер для ввода

int found = 0; // Флаг, найден ли ТС

int search\_by\_id = 0; // Флаг, поиск пр Id

int id = 0; // Переменная для хранения ID

char confirm; // Переменная для ответа пользователя

printf("\nDelete Vehicle\nEnter ID or model: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0';

char\* endptr;

id = strtol(input, &endptr, 10);

if (\*endptr == '\0') search\_by\_id = 1;

for (int i = 0; i < transport\_count; i++)

{

if ((search\_by\_id && (\*(transports + i)).id == id) ||

(!search\_by\_id && strcmp((\*(transports + i)).model, input) == 0))

{

found = 1;

printf("\nVehicle to Delete:\n");

printf("ID: %d\n", (\*(transports + i)).id);

printf("Type: %s\n", (\*(transports + i)).type == TRUCK ? "Truck" : "Passenger");

printf("Model: %s\n", (\*(transports + i)).model);

printf("Capacity: %d\n", (\*(transports + i)).capacity);

printf("Available: %d\n", (\*(transports + i)).available);

printf("Status: %s\n", (\*(transports + i)).current\_status);

if ((\*(transports + i)).booking\_count > 0)

{

printf("\nWARNING: This vehicle has %d active bookings!\n", (\*(transports + i)).booking\_count);

}

printf("\nAre you sure you want to delete this vehicle? (y/n): ");

scanf\_s(" %c", &confirm);

while (getchar() != '\n');

if (tolower(confirm) != 'y')

{

printf("Deletion cancelled\n");

return;

}

if ((\*(transports + i)).bookings)

{

free((\*(transports + i)).bookings);

}

for (int j = i; j < transport\_count - 1; j++)

{

transports[j] = transports[j + 1];

}

transport\_count--;

printf("Vehicle deleted successfully\n");

saveToFile();

return;

}

}

if (!found) printf("Vehicle not found!\n");

}

**ПРИЛОЖЕНИЕ З**

(обязательное)

Ведомость документов