МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Программирование

Отчет по курсовой работе на тему:

«Система учета автобусов для автовокзала»

Выполнил студент группы М3О-209Б-22

Кожевников А. В.

Принял:

Секретарев В. Е.

Москва 2023 г.

# Оглавление

[Оглавление 1](#_Toc154158721)

[Задание 2](#_Toc154158722)

[Структурные схемы алгоритмов 4](#_Toc154158723)

[Функция main 4](#_Toc154158724)

[Функция create\_stations 5](#_Toc154158725)

[Функция switch\_add 7](#_Toc154158726)

[Функция add\_stations 10](#_Toc154158727)

[Функция switch\_del 11](#_Toc154158728)

[Функция switch\_move 13](#_Toc154158729)

[Функция makeStation 15](#_Toc154158730)

[Функция DeleteStation 16](#_Toc154158731)

[Функция findElemStation 18](#_Toc154158732)

[Функция find\_id\_station 19](#_Toc154158733)

[Функция find\_route\_stationID 20](#_Toc154158734)

[Тестирование программы 22](#_Toc154158735)

[Тестирование некорректных режимов 22](#_Toc154158736)

[Тестирование корректных режимов 29](#_Toc154158737)

[Вывод 39](#_Toc154158738)

# Задание

**Схема организации данных**

Линейные и кольцевые двусвязные списки

**Тема курсовой работы**

Система учета автобусов для автовокзала: название автовокзала, марка автобуса, количество проданных билетов, расчетное количество пассажиров, цена билета, номер рейса, время отправления, пункт назначения, ФИО водителя.

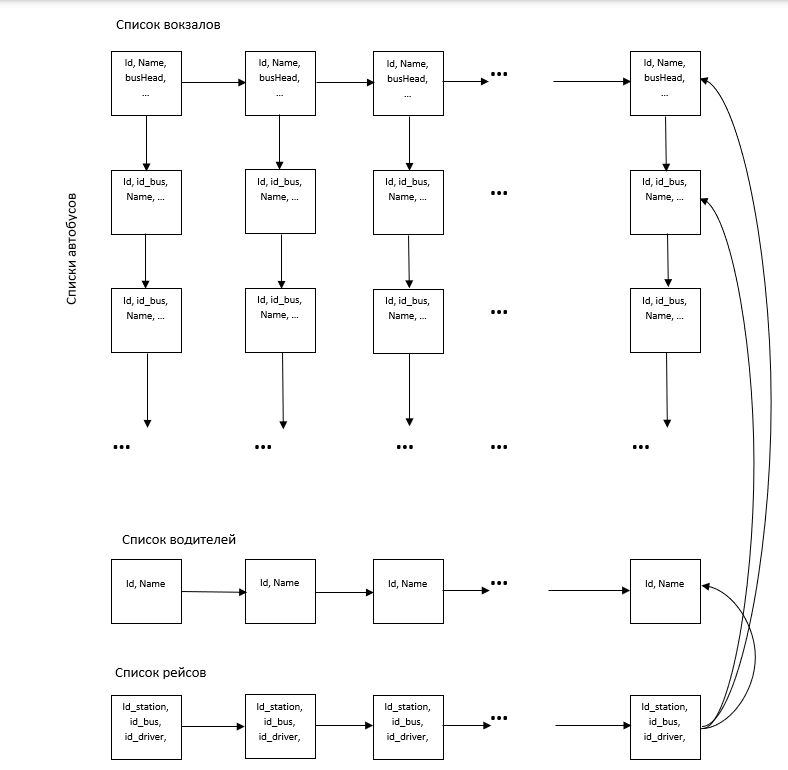
В каждой работе предусмотреть функции добавления, удаления элементов, поиска, печати, перемещение в прямом и обратном направлениях (для двусвязного списка), обеспечить уникальность значений полей и защиту от одновременного использования одного ресурса, где это необходимо. Возможно несколько узлов с одинаковыми значениями полей. Обеспечить выдачу запросов по любым полям структуры.

Создать списочную структуру, заполнить достаточным количеством данных, провести тестирование. Инициализацию структур произвести данными из файла. В отчете представить разработанные структуры данных, описать алгоритмы работы с ними для каждой функции.

Проект должен состоять из трех частей: заголовочные файлы \*.h, файлы с реализациями функций \*.cpp, файл основной программы\*.cpp.

Проект должен обеспечивать простоту и удобство масштабирования поставленной задачи.

# Схема списков



# Описание структур

Station

1. Назначение: Структура, содержащая данные вокзале.

2. Прототип структуры:

struct Station // Двунаправленный кольцевой АВТОВОКЗАЛЫ

{

Station\* prev;

Station\* next;

int id; // ID

string name; // Назв автовокзала

Bus\* busHead = NULL; // Голова списка автобусов для данного вокзала

Bus\* busEnd = NULL; // Хвост списка автобусов

};

3. Описание полей:

|  |  |  |
| --- | --- | --- |
| Имя параметра | Тип параметра | Назначение |
| next | Station\* | Указатель на след. эл-т |
| prev | Station\* | Указатель на пред. Элемент |
| id | int | Id данного вокзала |
| name | string | Название данного вокзала |
| busHead | Bus\* | Указатель на первый элемент списка автобусов, относящихся к данному вокзалу |
| busEnd | Bus\* | Указатель на последний элемент списка автобусов, относящихся к данному вокзалу |

Bus

1. Назначение:

Структура, содержащая данные о марках (автобусах), относящихся к конкретному вокзалу

2. Прототип структуры:

struct Bus // Двунаправленный кольцевой АВТОБУСЫ

{

Bus\* prev;

Bus\* next;

int id; // ID автовокзала

int id\_bus; // ID автобуса

string name; // Назв марки

};

3. Описание полей:

|  |  |  |
| --- | --- | --- |
| Имя параметра | Тип параметра | Назначение |
| next | Bus\* | Указатель на след. эл-т |
| prev | Bus\* | Указатель на пред. элемент |
| id | int | Id вокзала, к которому относится данный автобус |
| Id\_bus | int | Id данного автобуса (уникальный) |
| name | string | Название марки (уникальна в данном списке) |

Driver

1. Назначение:

Структура, содержащая данные водителях

2. Прототип структуры:

struct Driver // Водитель (линейный)

{

Driver\* next;

string name; // ФИО

int id; // ID водителя

};

3. Описание полей:

|  |  |  |
| --- | --- | --- |
| Имя параметра | Тип параметра | Назначение |
| next | Driver\* | Указатель на след. эл-т |
| id | int | Id данного водителя |
| name | string | ФИО водителя |

Route

1. Назначение:

Структура, содержащая данные о рейсах

2. Прототип структуры:

struct Route { // Линейный (рейсы)

Route\* next;

int id\_station; // ID автовокзала

int id\_bus; // ID автобуса

int id\_driver; // ID водителя

int route\_number; // Номер рейса

string time; // Время отправления

int tickets; // Количество проданных билетов

int passengers; // Количество пассажиров

string end\_route; // Место назначения

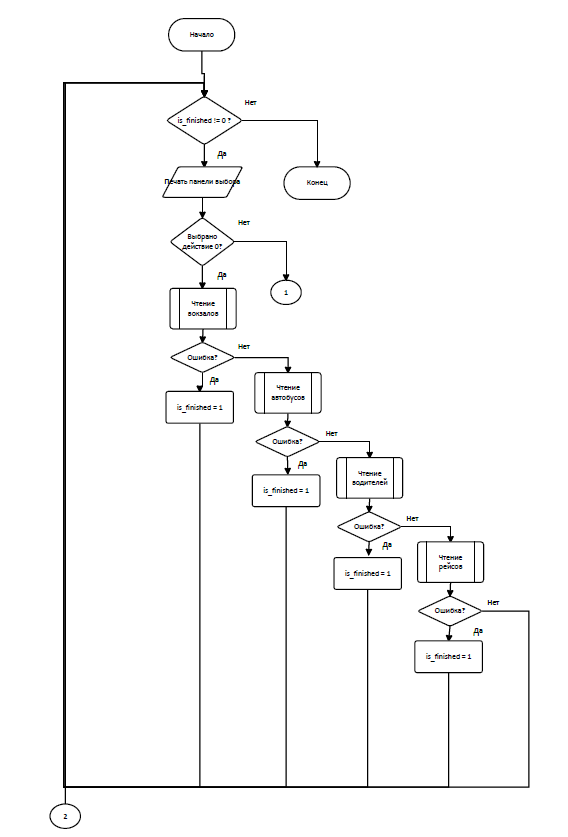
};

3. Описание полей:

|  |  |  |
| --- | --- | --- |
| Имя параметра | Тип параметра | Назначение |
| next | Route\* | Указатель на следующий элемент |
| Id\_station | int | Id вокзала |
| Id\_bus | int | Id автобуса |
| Id\_driver | int | Id водителя |
| Route\_number | int | Номер рейса (уникален) |
| time | string | Время отправления |
| tickets | int | Количество проданных билетов |
| passengers | int | Количество пассажиров |
| End\_route | string | Место назначения |

# Структурные схемы алгоритмов

## Функция main



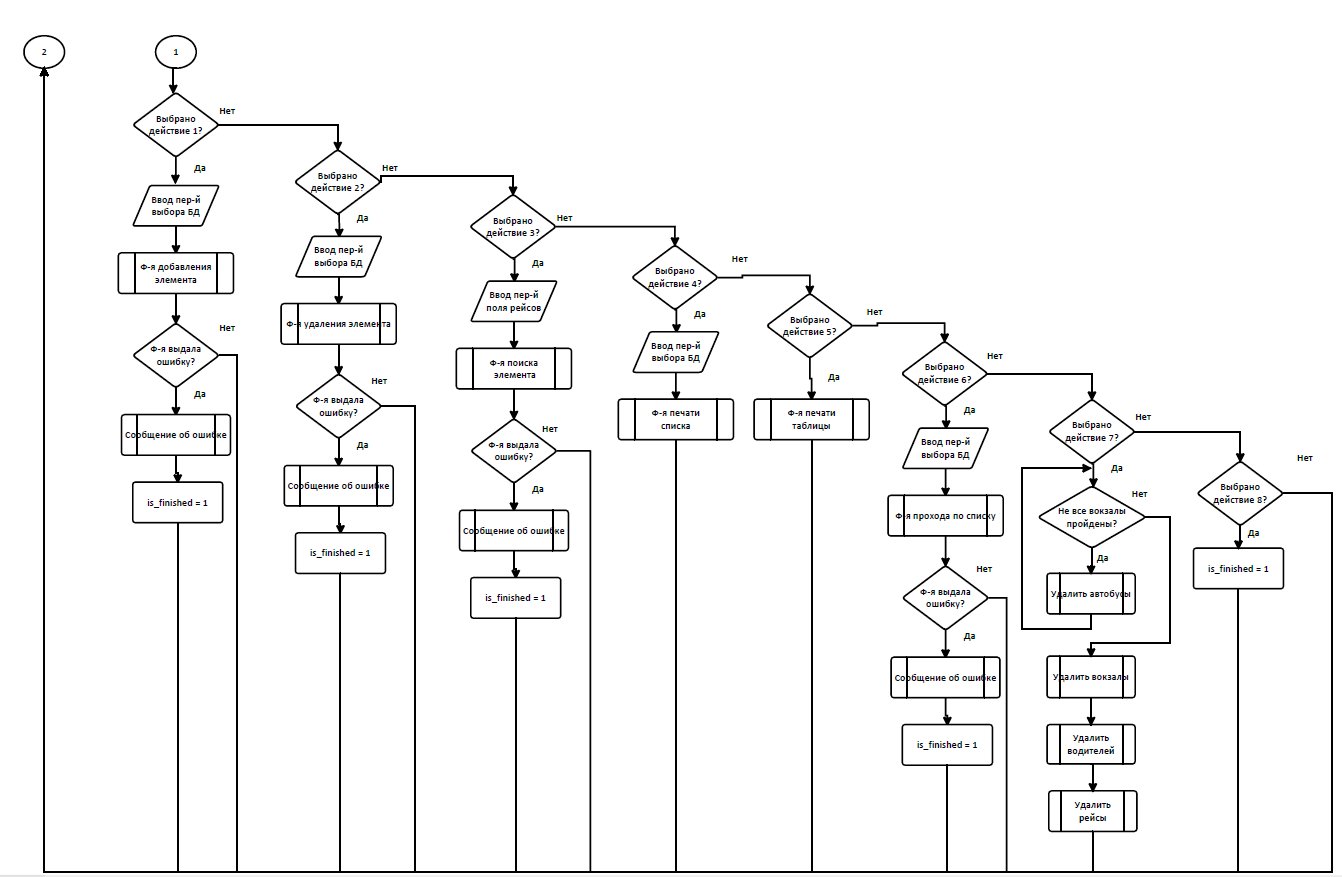


Рисунок 1. Структурная схема алгоритма функции main.

## Функция create\_stations

1. Назначение:

Создание списка вокзалов, представленных в файле

2. Прототип функции:

int create\_stations(const char\* fname, Station\*& head, Station\*& end)

3. Обращение к функции:

code = create\_stations(fstations, station\_head, station\_end)

4. Блок-схема:

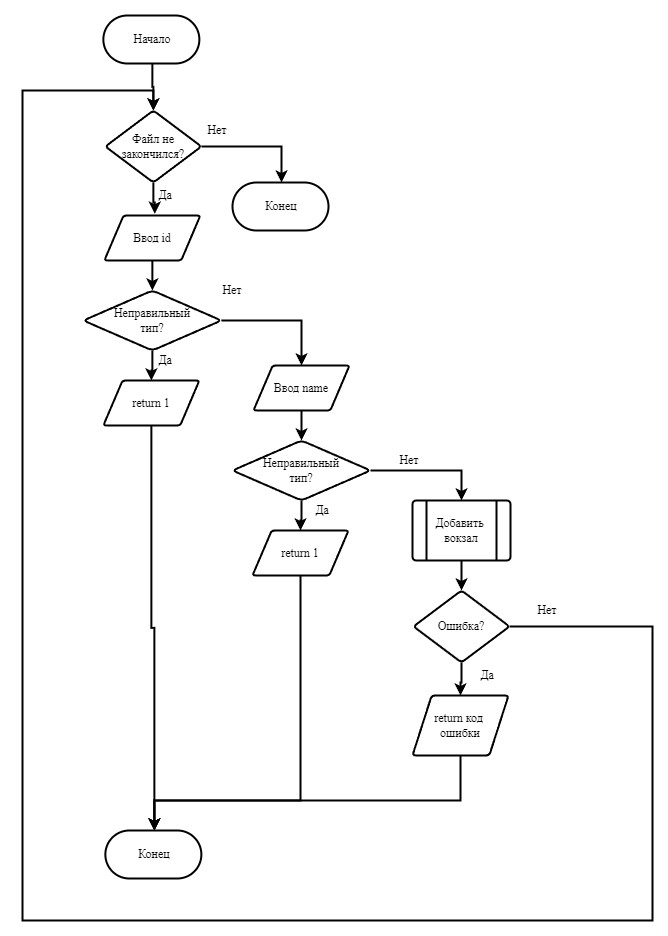


Рисунок 2. Структурная схема алгоритма функции create\_stations

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| head | Station\* | Указатель на первый элемент списка вокзалов | Вход |
| end | Station\* | Указатель на последний элемент списка вокзалов | Вход |
| fname | Const char\* | Имя файла | Вход |
| is\_mistake | int | Код ошибки добавления вокзала | Выход |

## Функция switch\_add

1. Назначение:

Запрос пользователю на выбор списка для добавления нового элемента

2. Прототип функции:

int switch\_add(Station\*& s\_head, Station\*& s\_end, Driver\*& d\_head, Driver\*& d\_end, Route\*& r\_head, Route\*& r\_end)

3. Обращение к функции:

code = switch\_add(station\_head, station\_end, driver\_head, driver\_end, route\_head, route\_end)

4. Блок-схема:

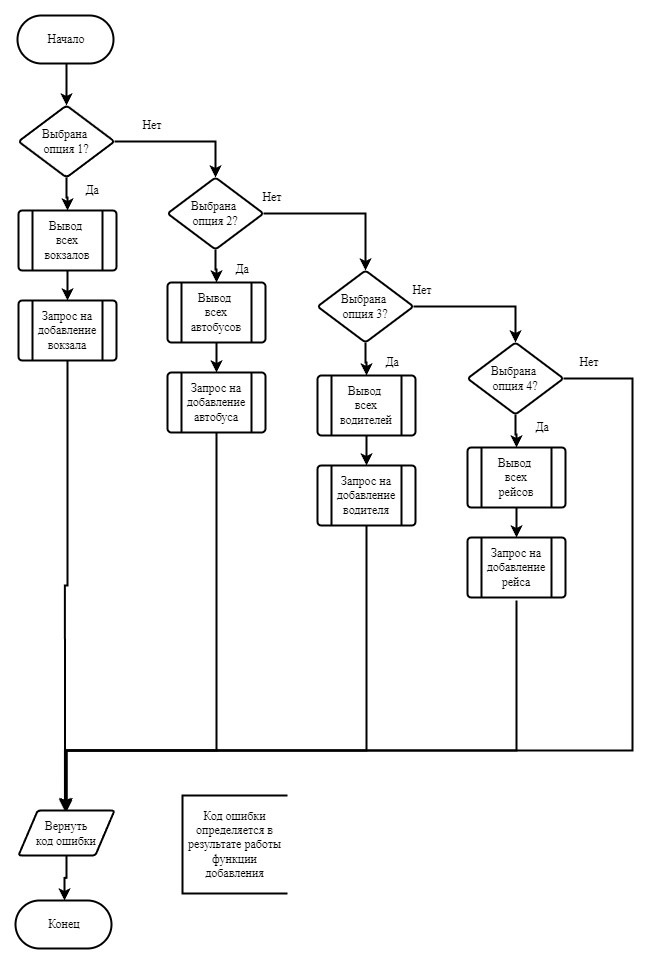


Рисунок 3. Структурная схема алгоритма функции switch\_add

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| s\_head | Station\* | Указатель на первый элемент списка вокзалов | Вход |
| s\_end | Station\* | Указатель на последний элемент списка вокзалов | Вход |
| d\_head | Driver\* | Указатель на первый элемент списка водителей | Вход |
| d\_end | Driver\* | Указатель на последний элемент списка водителей | Вход |
| r\_head | Route\* | Указатель на первый элемент списка рейсов | Вход |
| r\_end | Route\* | Указатель на последний элемент списка рейсов | Вход |

## Функция add\_stations

1. Назначение:

Запрос пользователю на добавление нового вокзала

2. Прототип функции:

int add\_stations(Station\*& head, Station\*& end)

3. Обращение к функции:

mistake\_code = add\_stations(s\_ head, s\_end)

4. Блок-схема:

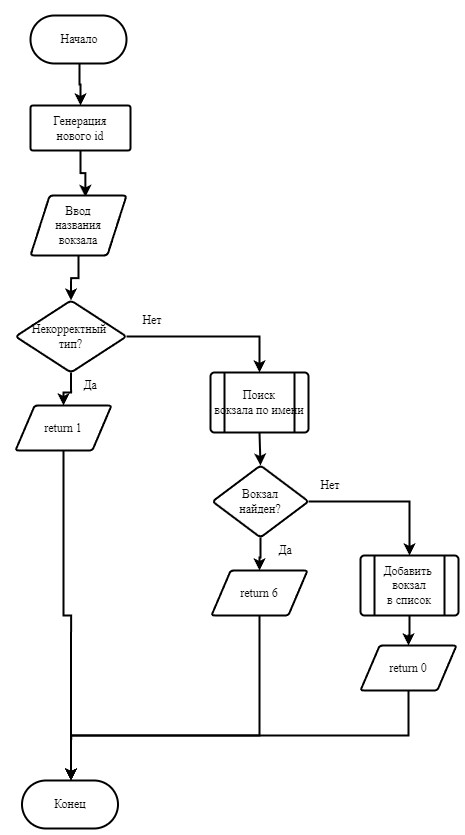


Рисунок 4. Структурная схема алгоритма функции add\_stations

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| head | Station\* | Указатель на первый элемент списка вокзалов | Вход |
| end | Station\* | Указатель на последний элемент списка вокзалов | Вход |
| loc\_choice | int | Глобальная переменная выбора опции | Вход |

## Функция switch\_del

1. Назначение:

Запрос пользователю на выбор списка для удаления элемента

2. Прототип функции:

int switch\_del(Station\*& s\_head, Station\*& s\_end, Driver\*& d\_head, Driver\*& d\_end, Route\*& r\_head, Route\*& r\_end)

3. Обращение к функции:

code = switch\_del(station\_head, station\_end, driver\_head, driver\_end, route\_head, route\_end)

4. Блок-схема:

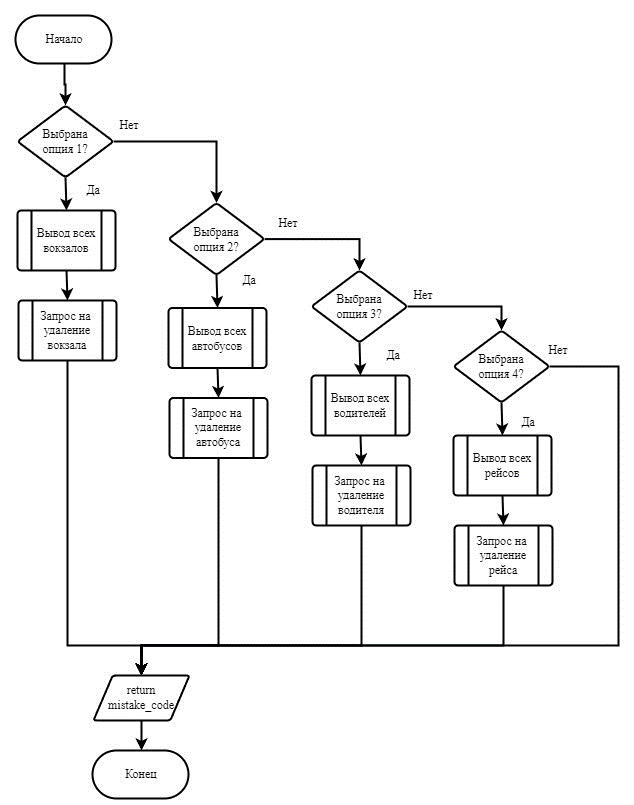


Рисунок 5. Структурная схема алгоритма функции switch\_del

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| s\_head | Station\* | Указатель на первый элемент списка вокзалов | Вход |
| s\_end | Station\* | Указатель на последний элемент списка вокзалов | Вход |
| d\_head | Driver\* | Указатель на первый элемент списка водителей | Вход |
| d\_end | Driver\* | Указатель на последний элемент списка водителей | Вход |
| r\_head | Route\* | Указатель на первый элемент списка рейсов | Вход |

## Функция switch\_move

1. Назначение:

Запрос пользователю на выбор списка для перемещения

2. Прототип функции:

int switch\_move(Station\* s\_head, Station\*& curr\_station, Bus\*& curr\_bus)

3. Обращение к функции:

code = switch\_ move(station\_head, curr\_station, curr\_bus)

4. Блок-схема:

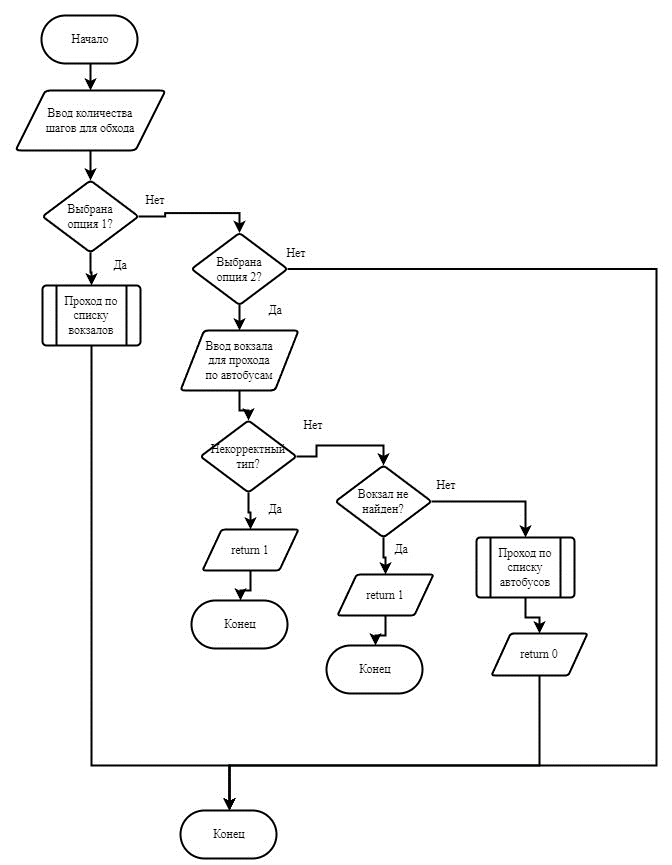


Рисунок 6. Структурная схема алгоритма функции switch\_move

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| s\_head | Station\* | Указатель на первый элемент списка вокзалов | Вход |
| curr\_station | Station\* | Указатель на последний посещенный вокзал | Вход |
| curr\_bus | Bus\* | Указатель на последний посещенный автобус | Вход |

## Функция makeStation

1. Назначение:

Добавление нового вокзала в список

2. Прототип функции:

int makeStation(int id, string name, Station\*& end, Station\*& head)

3. Обращение к функции:

is\_mistake = makeStation(id, name, end, head)

4. Блок-схема:

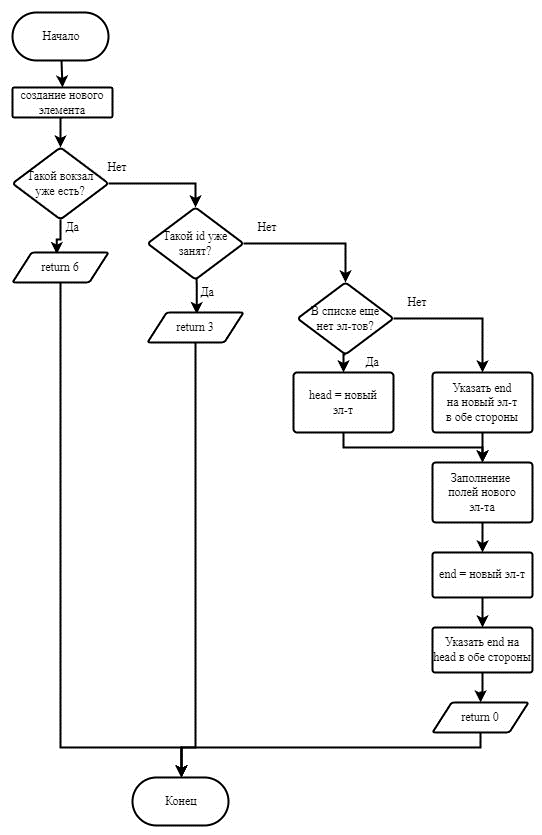


Рисунок 7. Структурная схема алгоритма функции makeStation

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| id | Int | id нового вокзала | Вход |
| name | string | Название нового вокзала | Вход |
| end | Station\* | Указатель на последний вокзал | Вход |
| head | Station\* | Указатель на первый вокзал | Вход |

## Функция DeleteStation

1. Назначение:

Удаление вокзала по его id

2. Прототип функции:

void DeleteStation(Station\*& head, Station\*& end, int id)

3. Обращение к функции:

DeleteStation(head, end, id)

4. Блок-схема:

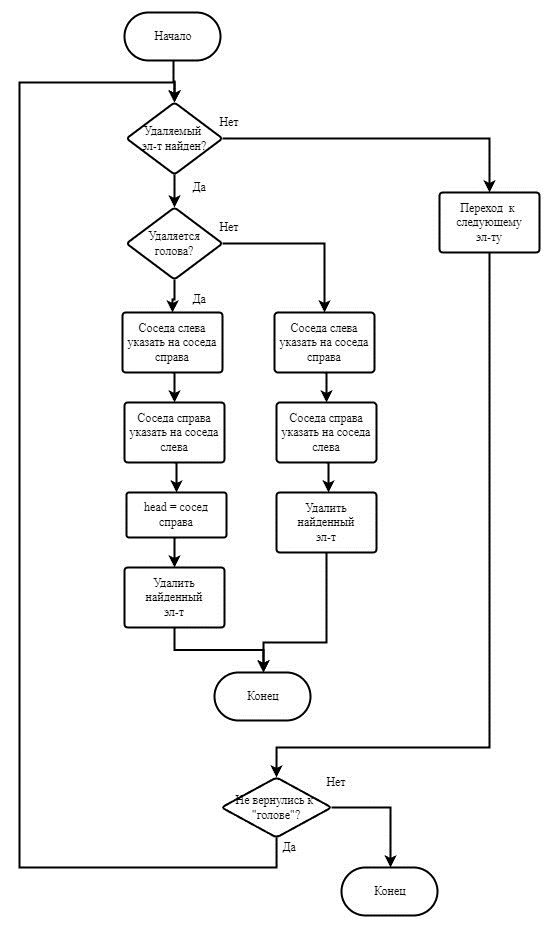


Рисунок 8. Структурная схема алгоритма функции makeStation

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| id | Int | id вокзала | Вход |
| end | Station\* | Указатель на последний вокзал | Вход |
| head | Station\* | Указатель на первый вокзал | Вход |

## Функция findElemStation

1. Назначение:

Поиск указателя на искомый вокзал по его id

2. Прототип функции:

Station\* findElemStation(int id, Station\* head)

3. Обращение к функции:

findElemStation(id, head)

4. Блок-схема:

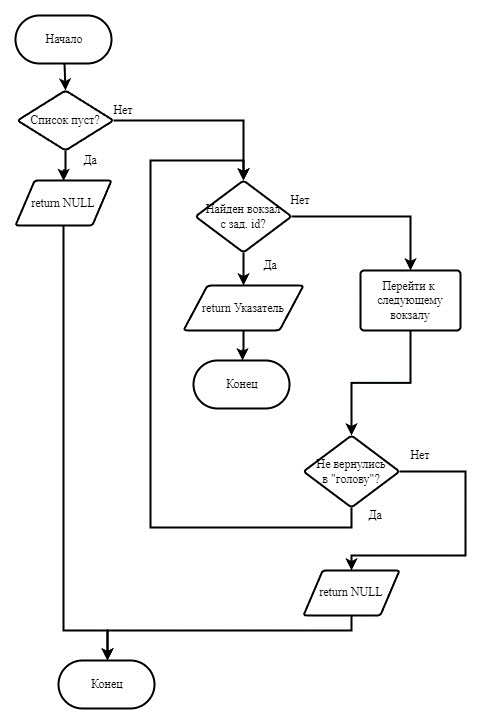


Рисунок 9. Структурная схема алгоритма функции findElemStation

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| id | Int | id вокзала | Вход |
| head | Station\* | Указатель на первый вокзал | Вход |
| ptr | Station\* | Указатель на найденный вокзал | Выход |

## Функция find\_id\_station

1. Назначение:

Поиск указателя на искомый вокзал по его названию

2. Прототип функции:

int find\_id\_station(string name, Station\* head)

3. Обращение к функции:

id = find\_id\_station(str\_filter, st\_head)

4. Блок-схема:

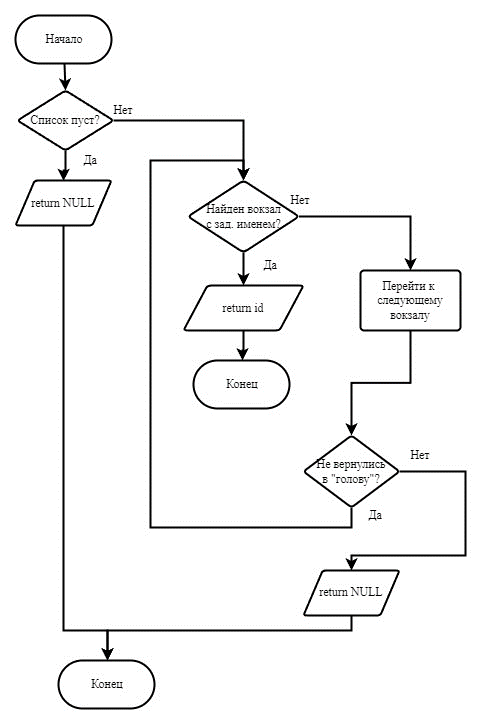


Рисунок 10. Структурная схема алгоритма функции find\_id\_station

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| name | string | название вокзала | Вход |
| head | Station\* | Указатель на первый вокзал | Вход |
| Id | int | Id найденного вокзала | Выход |

## Функция find\_route\_stationID

1. Назначение:

Поиск указателя на искомый вокзал, находящийся в одном из рейсов, по его id

2. Прототип функции:

Route\* find\_route\_stationID(int id\_station, Route\* head)

3. Обращение к функции:

found = find\_route\_stationID(id, route\_head)

4. Блок-схема:

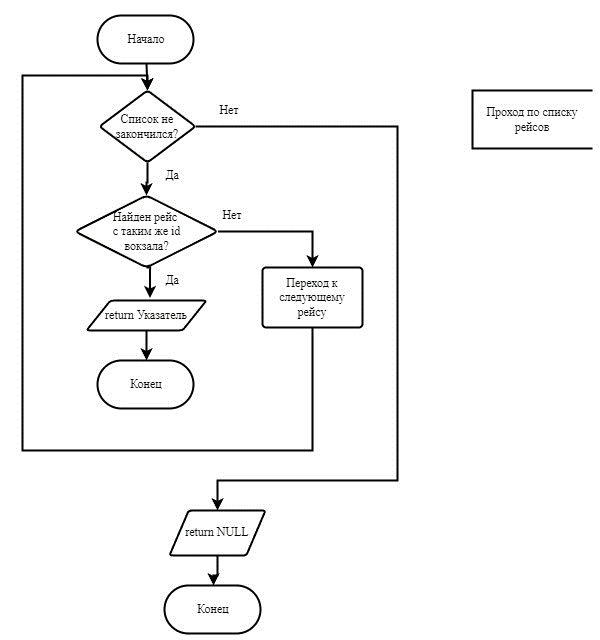
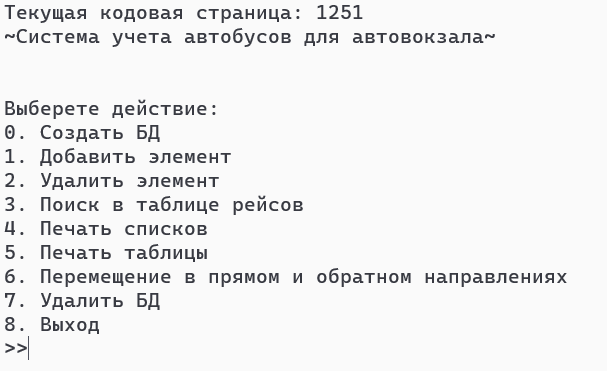


Рисунок 11. Структурная схема алгоритма функции find\_route\_stationID

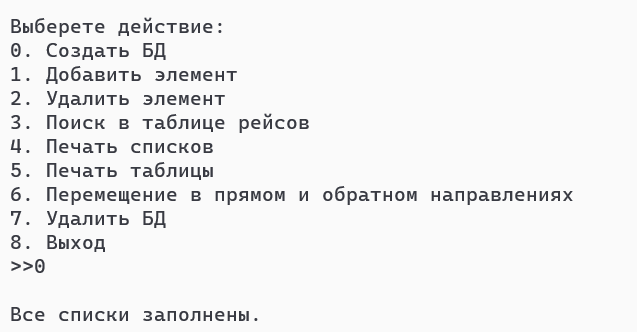
5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| ptr | Route\* | Указатель на рейс с искомым вокзалом | Выход |
| head | Route\* | Указатель на первый рейс | Вход |
| Id\_station | int | Id искомого вокзала | Вход |

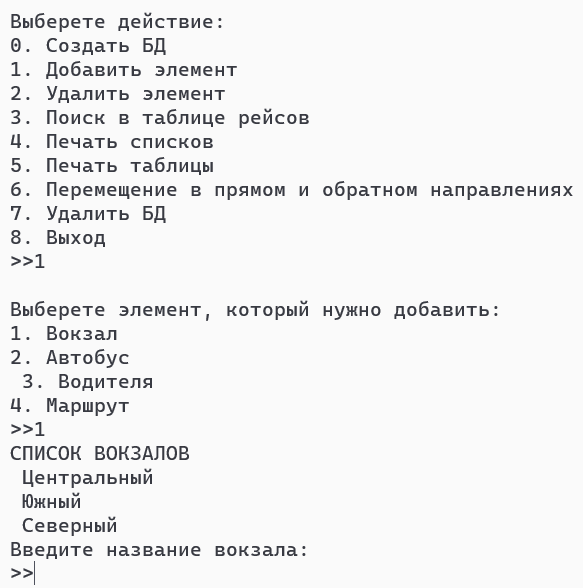
# Описание интерфейса



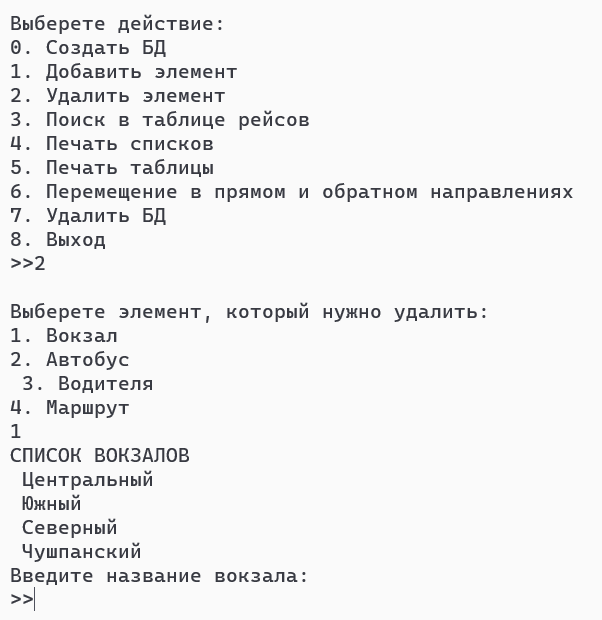
Опция 0 заполняет все списки данными из файлов



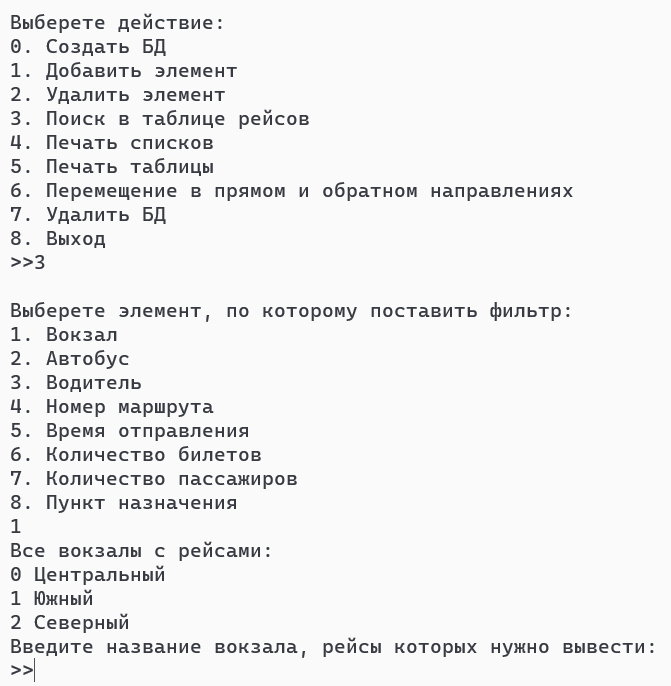
Опция 1 запускает функцию добавления элемента, предлагая предварительно выбрать список для добавления.



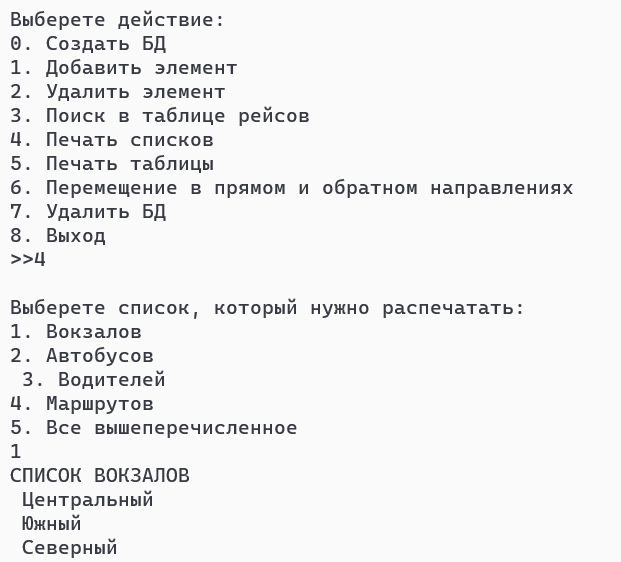
Опция 2 запускает функцию удаления элемента, предлагая предварительно выбрать список для удаления.



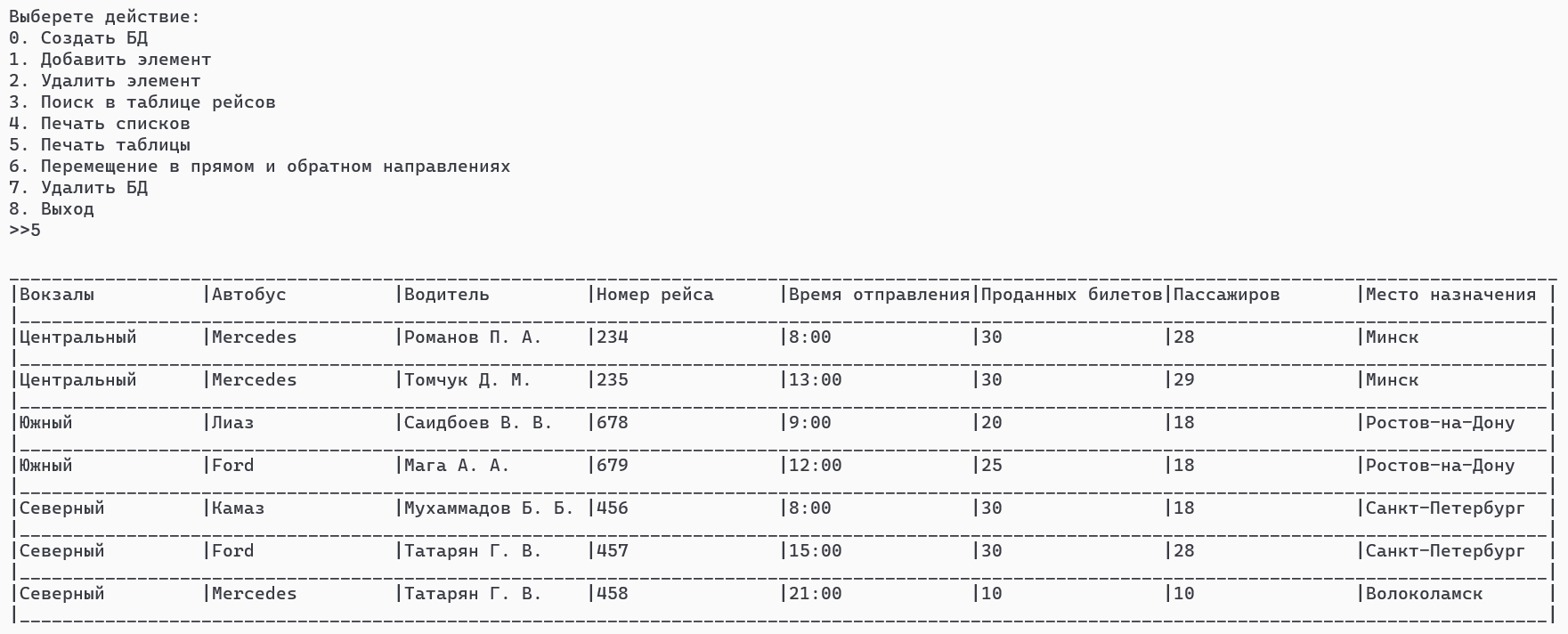
Опция 3 запускает функцию поиска элемента среди списка рейсов.



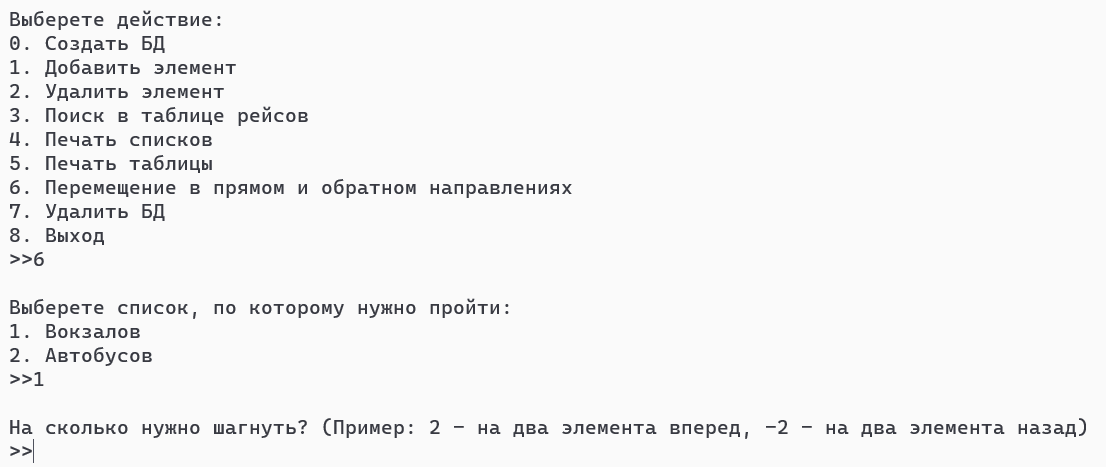
Опция 4 запускает функцию печати списков, предлагая предварительно выбрать сам список.



Опция 5 запускает функцию печати таблицы рейсов.



Опция 6 запускает функцию обхода кольцевых списков (вокзалов и автобусов) в прямом и обратном направлениях.



Опция 7 удаляет все списки.

Опция 8 совершает выход из программы.

# Код программы

Файл course\_job.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Программирование \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project type :Win64 Console Application \*

\*Project name\_bus :course\_job.sln \*

\*File name\_bus :course\_job.cpp \*

\*Language :CPP, MSVS 2022 \*

\*Programmers :Кожевников Артем Вадимович, М3О-209Б-22 \*

\*Modified By : \*

\*Created :06.12.2023 \*

\*Last revision:18.12.2023 \*

\*Comment : \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

#include <math.h>

#include "lists.h"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* П Р О Т О Т И П Ы Ф У Н К Ц И Й \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int create\_stations(const char\* fname, Station\*& head, Station\*& end); // Создание списка вокзалов

int create\_buses(const char\* fname, Station\* head); // Создание списков марок к каждому вокзалу

int create\_drivers(const char\* fname, Driver\*& head, Driver\*& end); // Создание списка рейсов

int create\_routes(const char\* fname, Route\*& head, Route\*& end); // Создание списка рейсов

// ФУНКЦИИ ДОБАВЛЕНИЯ

int switch\_add(Station\*& s\_head, Station\*& s\_end, Driver\*& d\_head, Driver\*& d\_end, Route\*& r\_head, Route\*& r\_end);

int add\_stations(Station\*& head, Station\*& end);

int add\_buses(Station\* head);

int add\_drivers(Driver\*& head, Driver\*& end);

int add\_routes(Route\*& head, Route\*& end, Station\* head\_station, Driver\* head\_driver);

// ФУНКЦИИ УДАЛЕНИЯ

int switch\_del(Station\*& s\_head, Station\*& s\_end, Driver\*& d\_head, Driver\*& d\_end, Route\*& r\_head, Route\*& r\_end);

int del\_station(Station\*& head, Station\*& end, Route\*& route\_head, Route\*& route\_end);

int del\_bus(Station\* head, Route\*& route\_head, Route\*& route\_end);

int del\_driver(Driver\*& head, Driver\*& end, Route\*& route\_head, Route\*& route\_end);

int del\_route(Route\*& head, Route\*& end);

// ФУНКЦИИ ПЕЧАТИ ТАБЛИЦЫ

void print\_table(Route\* head\_route, Station\* head\_station, Driver\* head\_driver); // Печать таблицы (проход по списку рейсов)

void print\_header(); // Печать шапки

void print\_raw(Route\* ptr, Station\* head\_station, Driver\* head\_driver); // Печать одной строчки

// ФУНКЦИИ ПЕЧАТИ

int switch\_print(Station\*& s\_head, Driver\*& d\_head, Route\*& r\_head);

// ФУНКЦИИ ПОИСКА

// Поиск по полю -> вывод в виде таблицы

int switch\_search(Route\* route\_head, Station\* st\_head, Driver\* dr\_head);

// ФУНКЦИИ ПЕРЕМЕЩЕНИЯ В ПРЯМОМ И ОБРАТНОМ НАПРАВЛЕНИЯХ

int switch\_move(Station\* s\_head, Station\*& curr\_station, Bus\*& curr\_bus);

// ОСТАЛЬНЫЕ ФУНКЦИИ

void PrintMess(int code); // Сообщение об ошибке

//------ГЛОБАЛЬНЫЕ КОНСТАНТЫ------//

// Некорректный тест №1

//const char\* fstations = "stations\_incorr1.txt";

//const char\* fbuses = "buses.txt";

//const char\* fdrivers = "drivers.txt";

//const char\* froutes = "routes.txt";

// Некорректный тест №2

//const char\* fstations = "stations.txt";

//const char\* fbuses = "buses\_incorr3.txt";

//const char\* fdrivers = "drivers.txt";

//const char\* froutes = "routes.txt";

// Некорректный тест №3

//const char\* fstations = "stations.txt";

//const char\* fbuses = "buses.txt";

//const char\* fdrivers = "drivers.txt";

//const char\* froutes = "routes\_incorr5.txt";

// Некорректный тест №4

//const char\* fstations = "stations.txt";

//const char\* fbuses = "buses.txt";

//const char\* fdrivers = "drivers\_incorr6.txt";

//const char\* froutes = "routes.txt";

// Некорректный тест №5-10

const char\* fstations = "stations.txt";

const char\* fbuses = "buses.txt";

const char\* fdrivers = "drivers.txt";

const char\* froutes = "routes.txt";

// Корректный тест №1-10

//const char\* fstations = "stations\_correct.txt";

//const char\* fbuses = "buses\_correct.txt";

//const char\* fdrivers = "drivers\_correct.txt";

//const char\* froutes = "routes\_correct.txt";

// Ширина столбцов для таблицы

const int table\_col = 18;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Г Л А В Н А Я П Р О Г Р А М М А \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

system("chcp 1251");

int choice = NULL; // Переменная выбора действия

bool is\_finished = 0; // закончили

bool are\_lists\_created = 0; // Создали ли БД

int code = 0; // Код ошибки

Station\* station\_head = NULL;

Station\* station\_end = NULL;

Driver\* driver\_head = NULL;

Driver\* driver\_end = NULL;

Route\* route\_head = NULL;

Route\* route\_end = NULL;

Station\* curr\_station = NULL; // Для перемещения в прямом и обратном направлениях (вокзал)

Bus\* curr\_bus = NULL; // Для перемещения в прямом и обратном направлениях (автобус)

Station\* ptr\_station; // Для удаления БД

cout << "~Система учета автобусов для автовокзала~" << endl;

while (!is\_finished)

{

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

cout << "\nВыберете действие: " << endl;

cout << "0. Создать БД " << endl;

cout << "1. Добавить элемент" << endl;

cout << "2. Удалить элемент" << endl;

cout << "3. Поиск в таблице рейсов" << endl;

cout << "4. Печать списков" << endl;

cout << "5. Печать таблицы" << endl; // Предусмотреть случаи добавления/удаления элементов

cout << "6. Перемещение в прямом и обратном направлениях" << endl;

cout << "7. Удалить БД" << endl;

cout << "8. Выход" << endl;

cout << ">>"; cin >> choice; cout << endl;

switch (choice)

{

case 0:

// Создание БД, с которой собираемся работать

code = create\_stations(fstations, station\_head, station\_end);

if (code)

{

PrintMess(code);

is\_finished = 1;

break;

}

curr\_station = station\_head; // Текущий вокзал - самый первый

code = create\_buses(fbuses, station\_head);

if (code)

{

PrintMess(code);

is\_finished = 1;

break;

}

curr\_bus = station\_head->busHead; // Текущий автобус - самый первый в списке у первого вокзала

code = create\_drivers(fdrivers, driver\_head, driver\_end);

if (code)

{

PrintMess(code);

is\_finished = 1;

break;

}

code = create\_routes(froutes, route\_head, route\_end);

if (code)

{

PrintMess(code);

is\_finished = 1;

break;

}

cout << "Все списки заполнены." << endl;

break;

case 1:

code = switch\_add(station\_head, station\_end, driver\_head, driver\_end, route\_head, route\_end);

if (code) // Ошибка

{

PrintMess(code);

}

break;

case 2:

code = switch\_del(station\_head, station\_end, driver\_head, driver\_end, route\_head, route\_end);

if (code) // Ошибка

{

PrintMess(code);

}

break;

case 3:

code = switch\_search(route\_head, station\_head, driver\_head);

if (code) // Ошибка

{

PrintMess(code);

}

break;

case 4:

switch\_print(station\_head, driver\_head, route\_head);

break;

case 5:

print\_table(route\_head, station\_head, driver\_head);

break;

case 6: // TODO Проверка на случай, если в вокзале вообще нет автобусов

code = switch\_move(station\_head, curr\_station, curr\_bus);

if (code) // Ошибка

{

PrintMess(code);

}

break;

case 7:

ptr\_station = station\_head;

// Удаление автобусов из всех вокзалов

do

{

delListBus(ptr\_station->busHead, ptr\_station->busEnd);

ptr\_station = ptr\_station->next;

} while (ptr\_station != station\_head);

delListStation(station\_head, station\_end); // Удаление всех вокзалов

del\_drivers(driver\_head);

del\_routes(route\_head);

case 8:

is\_finished = 1;

break;

default:

cout << "Некорректный ввод данных!" << endl;

break;

} // switch global

} // while

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Р Е А Л И З А Ц И Я Ф У Н К Ц И Й \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//------Создание БД из файла------//

int create\_stations(const char\* fname, Station\*& head, Station\*& end)

{

ifstream fin(fname);

int id;

string name; // Название вокзала

int is\_mistake = 0;

while (!fin.eof())

{

fin >> id;

if (fin.fail()) // DEBUG

{

fin.close();

return 1;

} // if

fin >> name;

if (fin.fail()) // DEBUG

{

fin.close();

return 1;

} // if

is\_mistake = makeStation(id, name, end, head);

if (is\_mistake) return is\_mistake; // DEBUG

} // while

fin.close();

return 0;

} // create\_stations

int create\_buses(const char\* fname, Station\* head)

{

ifstream fin(fname);

int id; // айди вокзала

int id\_bus; // айди автобуса

string name; // Название марки

int is\_mistake = 0;

Station\* ptr = head;

while (!fin.eof())

{

fin >> id;

if (fin.fail()) // DEBUG

{

fin.close();

return 1;

}

fin >> id\_bus;

if (fin.fail()) // DEBUG

{

fin.close();

return 1;

}

fin >> name;

if (fin.fail()) // DEBUG

{

fin.close();

return 1;

}

do

{

if (findElemBus(id\_bus, ptr->busHead)) return 3; // DEBUG (уникальность id автобусов)

ptr = ptr->next;

} while (ptr != head);

ptr = findElemStation(id, head); // Ищем вокзал

if (!ptr) // DEBUG

{

fin.close();

return 4;

}

Bus\* bus\_loc\_head = ptr->busHead; Bus\* bus\_loc\_end = ptr->busEnd;

is\_mistake = makeBus(id, id\_bus, name, bus\_loc\_end, bus\_loc\_head); // Добавляем автобус в вокзал

if (is\_mistake) return is\_mistake; // DEBUG (уникальность имени марки в одном вокзале)

if (!ptr->busHead) ptr->busHead = bus\_loc\_head; // Добавление головы в поле

ptr->busEnd = bus\_loc\_end; // Изменение поля конца списка

} // while

fin.close();

return 0;

} // create\_bus

int create\_drivers(const char\* fname, Driver\*& head, Driver\*& end) // Создание списка рейсов

{

ifstream fin(fname);

int id;

string full\_name; string name; string father\_name; // ФИО

int is\_mistake = 0;

while (!fin.eof())

{

fin >> id;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> full\_name;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> name;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> father\_name;

if (fin.fail())

{

fin.close();

return 1;

} // if

full\_name = full\_name + " " + name + " " + father\_name; // ФИО

is\_mistake = make\_driver(id, full\_name, end, head); // Добавить в список

if (is\_mistake) return is\_mistake;

} // while

fin.close();

return 0;

}

int create\_routes(const char\* fname, Route\*& head, Route\*& end)

{

int is\_mistake = 0;

ifstream fin(fname);

while (!fin.eof())

{

Route\* RouteLoc = new Route;

fin >> RouteLoc->id\_station;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> RouteLoc->id\_bus;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> RouteLoc->id\_driver;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> RouteLoc->route\_number;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> RouteLoc->time;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> RouteLoc->tickets;

if (fin.fail())

{

fin.close();

return 1;

} // if

fin >> RouteLoc->passengers;

if (fin.fail())

{

fin.close();

return 1;

} // if

if (RouteLoc->passengers > RouteLoc->tickets) return 7;

fin >> RouteLoc->end\_route;

if (fin.fail())

{

fin.close();

return 1;

} // if

is\_mistake = make\_route(RouteLoc, end, head);

if (is\_mistake) return is\_mistake;

} // while

fin.close();

return 0;

}

//------Добавление элементов------//

int switch\_add(Station\*& s\_head, Station\*& s\_end, Driver\*& d\_head, Driver\*& d\_end, Route\*& r\_head, Route\*& r\_end)

{

int loc\_choice = 0;

int mistake\_code = 0; // Код ошибки

if (!s\_head && !d\_head && !r\_head) return 8; // БД еще не создали

cout << "Выберете элемент, который нужно добавить: \n1. Вокзал\n2. Автобус\n 3. Водителя\n4. Маршрут\n>>";

cin >> loc\_choice;

switch (loc\_choice)

{

case 1:

printStation(s\_head); // Вывод всех вокзалов

mistake\_code = add\_stations(s\_head, s\_end);

break;

case 2:

print\_all\_buses(s\_head); // Вывод всех вокзалов и автобусов

mistake\_code = add\_buses(s\_head);

break;

case 3:

print\_drivers(d\_head); // Вывод всех водителей

mistake\_code = add\_drivers(d\_head, d\_end);

break;

case 4:

cout << "Все рейсы: " << endl;

print\_routes(r\_head, s\_head, d\_head);

mistake\_code = add\_routes(r\_head, r\_end, s\_head, d\_head);

break;

default:

cout << "Введено некорректное значение!";

break;

} // switch

return mistake\_code;

} // switch\_add

int add\_stations(Station\*& head, Station\*& end)

{

int id;

string name;

id = max\_station\_id(head) + 1; // генерируем новый id

cout << "Введите название вокзала: \n>>"; cin>>name;

if (cin.fail()) {

return 1; // Некорректное имя

}

else if (find\_id\_station(name, head) != -1) return 6; // Есть ли уже такой вокзал

makeStation(id, name, end, head);

return 0;

}

int add\_buses(Station\* head)

{

int new\_id = 0; int max\_id = 0;

string name\_bus;

string name\_station;

Station\* ptr = head;

Station\* station\_ptr = NULL; // Вокзал, к которому будет относиться автобус

int is\_mistake = 0;

cout << "Введите название вокзала, в который нужно добавить автобус: \n>>"; cin >> name\_station;

if (cin.fail()) {

return 1; // Некорректное имя

}

station\_ptr = findElemStation(find\_id\_station(name\_bus, head), head);

if (!station\_ptr) return 4; // Не найдена станция

cout << "\nВведите название марки: \n>>"; cin >> name\_bus;

// Определение нового id

do

{

max\_id = max\_bus\_id(ptr->busHead);

if (new\_id < max\_id) new\_id = max\_id;

} while (ptr != head);

Bus\* bus\_end = station\_ptr->busEnd;

is\_mistake = makeBus(station\_ptr->id, new\_id, name\_bus, bus\_end, station\_ptr->busHead);

if (is\_mistake) return is\_mistake;

station\_ptr->busEnd = bus\_end;

return 0;

}

int add\_drivers(Driver\*& head, Driver\*& end)

{

int id;

string name; string name1; string name2;

int is\_mistake = 0;

id = max\_driver\_id(head)+1;

cout << "Введите ФИО водителя: \n>>";

cin >> name;

if (cin.fail()) {

return 1; // Некорректное имя

}

cin >> name1;

if (cin.fail()) {

return 1; // Некорректное имя

}

cin >> name2;

if (cin.fail()) {

return 1; // Некорректное имя

}

name = name + " " + name1 + " " + name2;

is\_mistake = make\_driver(id, name, end, head);

if (is\_mistake) return is\_mistake;

return 0;

}

int add\_routes(Route\*& head, Route\*& end, Station\* head\_station, Driver\* head\_driver)

{

Route\* ptr\_route = new Route;

Station\* station\_ptr;

int is\_mistake = 0;

string name\_station;

string name\_bus;

string full\_name\_driver; string name\_driver; string name\_driver2;

cout << "Введите название существующего вокзала: "; cin >> name\_station;

if (cin.fail()) {

return 1; // Несоответствие типов данных!

}

ptr\_route->id\_station = find\_id\_station(name\_station, head\_station);

station\_ptr = findElemStation(ptr\_route->id\_station, head\_station);

if (!station\_ptr) return 4; // Не найден элемент

cout << "Введите название автобуса из данного вокзала: "; cin >> name\_bus;

if (cin.fail()) {

return 1; // Некорректнo

}

ptr\_route->id\_bus = find\_id\_bus(name\_bus, station\_ptr->busHead);

if (ptr\_route->id\_bus == -1) return 4; // Не найден элемент

cout << "Введите ФИО существующего водителя: ";

cin >> full\_name\_driver;

if (cin.fail()) {

return 1; // Некорректный route\_number

}

cin >> name\_driver;

if (cin.fail()) {

return 1; // Некорректный route\_number

} cin >> name\_driver2;

if (cin.fail()) {

return 1; // Некорректный route\_number

}

full\_name\_driver = full\_name\_driver + " " + name\_driver + " " + name\_driver2;

ptr\_route->id\_driver = find\_id\_driver(full\_name\_driver, head\_driver);

if (!ptr\_route->id\_driver) return 4; // Не найден элемент

cout << "Введите номер рейса: "; cin >> ptr\_route->route\_number;

if (cin.fail()) {

return 1; // Несоответствие типов данных!

}

else if (find\_route\_number(ptr\_route->route\_number, head)) return 5; // Если номер рейса уже забит

cout << "Введите время отправления: "; cin >> ptr\_route->time;

if (cin.fail()) {

return 1; // Несоответствие типов данных!

}

cout << "Введите количество проданных билетов: "; cin >> ptr\_route->tickets;

if (cin.fail()) {

return 1; // Несоответствие типов данных!

}

cout << "Введите количество пассажиров: "; cin >> ptr\_route->passengers;

if (cin.fail()) {

return 1; // Несоответствие типов данных!

}

if (ptr\_route->passengers > ptr\_route->tickets) return 7;

cout << "Введите место назначения: "; cin >> ptr\_route->end\_route;

if (cin.fail()) {

return 1; // Несоответствие типов данных!

}

is\_mistake = make\_route(ptr\_route, end, head);

if (is\_mistake) return is\_mistake;

return 0;

}

//------Удаление элементов------//

int switch\_del(Station\*& s\_head, Station\*& s\_end, Driver\*& d\_head, Driver\*& d\_end, Route\*& r\_head, Route\*& r\_end)

{

Station\* ptr = s\_head;

int mistake\_code = 0;

int loc\_choice = 0;

if (!s\_head && !d\_head && !r\_head) return 8; // БД еще не создали

cout << "Выберете элемент, который нужно удалить: \n1. Вокзал\n2. Автобус\n 3. Водителя\n4. Маршрут\n";

cin >> loc\_choice;

switch (loc\_choice)

{

case 1:

printStation(s\_head); // Вывод всех вокзалов

mistake\_code = del\_station(s\_head, s\_end, r\_head, r\_end);

break;

case 2:

print\_all\_buses(s\_head); // Вывод всех вокзалов и автобусов

mistake\_code = del\_bus(s\_head, r\_head, r\_end);

break;

case 3:

print\_drivers(d\_head); // Вывод всех водителей с рейсами

mistake\_code = del\_driver(d\_head, d\_end, r\_head, r\_end);

break;

case 4:

cout << "Все рейсы: " << endl;

print\_routes(r\_head, s\_head, d\_head);

mistake\_code = del\_route(r\_head, r\_end);

break;

default:

cout << "Введено некорректное значение!";

break;

} // switch

return mistake\_code;

} // switch\_del

int del\_station(Station\*& head, Station\*& end, Route\*& route\_head, Route\*& route\_end)

{

int id;

string name;

Station\* ptr;

Route\* route\_ptr;

cout << "Введите название вокзала: \n>>"; cin >> name;

if (cin.fail()) {

return 1; // Некорректный id

}

id = find\_id\_station(name, head);

if (id == -1) return 4; // Если такого вокзала нет

ptr = findElemStation(id, head);

// Удаляем рейсы с таким вокзалом

route\_ptr = find\_route\_stationID(id, route\_head);

while (route\_ptr)

{

delete\_route(route\_head, route\_end, route\_ptr->route\_number); // Удаляем по номеру маршрута

route\_ptr = find\_route\_stationID(id, route\_head); // Ищем еще

} // Удаляем все рейсы

delListBus(ptr->busHead, ptr->busEnd); // Удаляем автобусы из вокзала

DeleteStation(head, end, id); // Удалить вокзал

return 0;

}

int del\_bus(Station\* head, Route\*& route\_head, Route\*& route\_end)

{

int id; // ID автобуса

string name\_st;

string name\_bus;

Station\* ptr; // указатель на вокзал с этим автобусом

Route\* r\_ptr; // Рейс, в котором наверное есть автобус

cout << "Введите название станции, из которой удалить автобус: \n>>"; cin >> name\_st;

ptr = findElemStation(find\_id\_station(name\_st, head), head);

if (!ptr) return 4; // Если такого вокзала нет

cout << "Введите название марки: \n>>"; cin >> name\_bus;

id = find\_id\_bus(name\_bus, ptr->busHead);

if (cin.fail()) {

return 1; // Некорректный name\_bus

}

else if (id==-1) return 4; // Такого автобуса нет

// Удаляем рейсы, найдя указатель функцией поиска по полю id автобуса, по полю route\_number

r\_ptr = find\_route\_busID(id, route\_head);

while (r\_ptr)

{

if (r\_ptr) delete\_route(route\_head, route\_end, r\_ptr->route\_number);

r\_ptr = find\_route\_busID(id, route\_head);

}

// Удаляем, не забывая изменить поля

Bus\* bus\_head = ptr->busHead;

Bus\* end = ptr->busEnd;

DeleteBus(bus\_head, end, id);

ptr->busHead = bus\_head;

ptr->busEnd = end;

return 0;

}

int del\_driver(Driver\*& head, Driver\*& end, Route\*& route\_head, Route\*& route\_end)

{

int id;

string name;

Driver\* ptr;

cout << "Введите ФИО водителя: \n>>"; cin >> name;

if (cin.fail()) {

return 1; // Некорректный id

}

id = find\_id\_driver(name, head);

if (!find\_driver(id, head)) return 4; // Если такого нет

// Удаляем рейс, найдя указатель функцией поиска по полю id водителя, по полю route\_number

delete\_route(route\_head, route\_end, find\_route\_driverID(id, route\_head)->route\_number);

// Удаляем из списка водителей

delete\_driver(head, end, id);

return 0;

}

int del\_route(Route\*& head, Route\*& end)

{

int route\_number; Route\* ptr;

cout << "Введите номер удаляемого маршрута: \n>>"; cin >> route\_number;

if (cin.fail()) {

return 1; // Некорректный route\_number

}

ptr = find\_route\_number(route\_number, head);

if (!ptr) return 4; // Если такого нет

delete\_route(head, end, route\_number);

return 0;

}

//------Поиск элемента------//

int switch\_search(Route\* route\_head, Station\* st\_head, Driver\* dr\_head)

{

int loc\_choice = 0;

Route\* found; // Элемент, найденный по фильтру

Station\* st\_ptr = st\_head; // Проход для автобусов

string str\_filter;

int int\_filter;

int id; // Найденный id

bool found\_flag = 0; // Флаг: найден ли элемент

if (!st\_head && !dr\_head && !route\_head) return 8; // Не создали БД

cout << "Выберете элемент, по которому поставить фильтр: \n1. Вокзал\n2. Автобус\n3. Водитель\n4. Номер маршрута\n5. Время отправления\n6. Количество билетов\n7. Количество пассажиров\n8. Пункт назначения\n";

cin >> loc\_choice;

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

switch (loc\_choice)

{

case 1: // По вокзалам

print\_stations\_in\_route(route\_head, st\_head); // Вывод всех вокзалов с рейсами

cout << "Введите название вокзала, рейсы которых нужно вывести: \n>>"; cin >> str\_filter;

if (cin.fail()) return 1;

else if (find\_id\_station(str\_filter, st\_head) == -1) return 4; // Если такого вокзала просто нет

id = find\_id\_station(str\_filter, st\_head); // id ищем по списку вокзалов

found = find\_route\_stationID(id, route\_head); // ищем рейс с таким вокзалом

print\_header();

while (found) // Идем до конца

{

print\_raw(found, st\_head, dr\_head);

found = find\_route\_stationID(id, found->next); // Продвигаемся по списку РЕЙСОВ и ищем еще

} // while

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

case 2: // По автобусам

print\_buses\_in\_route(route\_head, st\_head); // Все автобусы в рейсах

cout << "Введите название автобуса, рейсы с которым нужно вывести: \n>>"; cin >> str\_filter;

if (cin.fail()) return 1;

print\_header();

do // Ищем по каждому вокзалу

{ // Ищем автобус в вокзале st\_ptr

found = route\_head;

id = find\_id\_bus(str\_filter, st\_ptr->busHead); // Вернет -1, если автобус в данном вокзале не найден

found = find\_route\_busID(id, route\_head); // Ищем автобус такого вокзала в списке рейсов

while (found && id != -1) // Идем до конца

{

found\_flag = 1;

print\_raw(found, st\_head, dr\_head);

found = find\_route\_busID(id, found->next); // Продвигаемся по списку и ищем еще

} // while

st\_ptr = st\_ptr->next;

} while (st\_ptr!=st\_head);

if (!found\_flag) return 4; // Такого автобуса просто нет

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

case 3:

print\_drivers\_in\_route(route\_head, dr\_head); // Все водители в рейсах

cout << "Введите ФИО водителя, рейсы которого нужно вывести: \n>>"; cin >> str\_filter;

if (cin.fail()) return 1;

else if (!find\_id\_driver(str\_filter, dr\_head)) return 4; // Если такого вокзала просто нет

id = find\_id\_driver(str\_filter, dr\_head);

found = find\_route\_driverID(id, route\_head);

print\_header();

while (found) // Идем до конца

{

print\_raw(found, st\_head, dr\_head);

found = find\_route\_driverID(id, found->next); // Продвигаемся по списку и ищем еще

} // while

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

case 4: // number

print\_routes(route\_head, st\_head, dr\_head); // Все рейсы

cout << "Введите номер рейса который нужно вывести: \n>>"; cin >> int\_filter;

if (cin.fail()) return 1;

found = find\_route\_number(int\_filter, route\_head);

if (!found) return 4; // Если такого вокзала просто нет

print\_header();

print\_raw(found, st\_head, dr\_head);

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

case 5: // time\_dep

print\_table(route\_head, st\_head, dr\_head); // Все существующие рейсы

cout << "Введите время рейсов для вывода: \n>>"; cin >> str\_filter;

if (cin.fail()) return 1;

else if (!find\_route\_time\_dep(str\_filter, route\_head)) return 4; // Если такого времени просто нет

found = find\_route\_time\_dep(str\_filter, route\_head);

print\_header();

while (found) // Идем до конца

{

print\_raw(found, st\_head, dr\_head);

found = find\_route\_time\_dep(str\_filter, found->next); // Продвигаемся по списку и ищем еще

} // while

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

case 6: // tickets

print\_table(route\_head, st\_head, dr\_head); // Все существующие рейсы

cout << "Введите количество билетов в рейсе для вывода: \n>>"; cin >> int\_filter;

if (cin.fail()) return 1;

else if (!find\_route\_tickets(int\_filter, route\_head)) return 4; // Если такого времени просто нет

found = find\_route\_tickets(int\_filter, route\_head);

print\_header();

while (found) // Идем до конца

{

print\_raw(found, st\_head, dr\_head);

found = find\_route\_tickets(int\_filter, found->next); // Продвигаемся по списку и ищем еще

} // while

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

case 7: // pass

print\_table(route\_head, st\_head, dr\_head); // Все существующие рейсы

print\_routes(route\_head, st\_head, dr\_head); // Все рейсы

cout << "Введите количество пассажиров в рейсе для вывода: \n>>"; cin >> int\_filter;

if (cin.fail()) return 1;

else if (!find\_route\_pass(int\_filter, route\_head)) return 4; // Если такого времени просто нет

found = find\_route\_pass(int\_filter, route\_head); print\_header();

while (found) // Идем до конца

{

print\_raw(found, st\_head, dr\_head);

found = find\_route\_pass(int\_filter, found->next); // Продвигаемся по списку и ищем еще

} // while

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

case 8: // end

print\_table(route\_head, st\_head, dr\_head); // Все существующие рейсы

cout << "Введите пункт назначения рейсов для вывода: \n>>"; cin >> str\_filter;

if (cin.fail()) return 1;

else if (!find\_route\_end(str\_filter, route\_head)) return 4; // Если такого времени просто нет

found = find\_route\_end(str\_filter, route\_head);

print\_header();

while (found) // Идем до конца

{

print\_raw(found, st\_head, dr\_head);

found = find\_route\_end(str\_filter, found->next); // Продвигаемся по списку и ищем еще

} // while

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

break;

default:

cout << "Введено некорректное значение!" << endl;

break;

} // switch

return 0;

} //

//------Печать списков------//

int switch\_print(Station\*& s\_head, Driver\*& d\_head, Route\*& r\_head)

{

int loc\_choice = 0;

if (!s\_head && !d\_head && !r\_head) return 8;

cout << "Выберете список, который нужно распечатать: \n1. Вокзалов\n2. Автобусов\n 3. Водителей\n4. Маршрутов\n5. Все вышеперечисленное\n";

cin >> loc\_choice;

switch (loc\_choice)

{

case 1: // Вокзалов

printStation(s\_head);

break;

case 2: // Автобусов

print\_all\_buses(s\_head);

break;

case 3: // Водителей

print\_drivers(d\_head);

break;

case 4: // Маршрутов

print\_routes(r\_head, s\_head, d\_head);

break;

case 5: // Все вышеперечисленное

printStation(s\_head);

print\_all\_buses(s\_head);

print\_drivers(d\_head);

print\_routes(r\_head, s\_head, d\_head);

break;

default:

cout << "Введено некорректное значение!"<<endl;

break;

} // switch 4

return 0;

}

//------Таблица------//

void print\_table(Route\* head\_route, Station\* head\_station, Driver\* head\_driver)

{

Station\* station\_ptr = NULL;

Bus\* bus\_ptr;

Driver\* driver\_ptr;

// ШАПКА

print\_header();

// Проход по списку рейсов

Route\* route\_ptr = head\_route;

while (route\_ptr)

{

print\_raw(route\_ptr, head\_station, head\_driver);

route\_ptr = route\_ptr->next;

}

// Подчеркиваем

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

}

void print\_header() // ШАПКА

{

cout << '\_' << setfill('\_') << setw(table\_col \* 8) << '\_' << endl;

cout << '|' << "Вокзалы" << setfill(' ') << setw(table\_col - 7)

<< '|' << "Автобус" << setfill(' ') << setw(table\_col - 7)

<< '|' << "Водитель" << setfill(' ') << setw(table\_col - 8)

<< '|' << "Номер рейса" << setfill(' ') << setw(table\_col - 11)

<< '|' << "Время отправления" << setfill(' ') << setw(table\_col - 17)

<< '|' << "Проданных билетов" << setfill(' ') << setw(table\_col - 17)

<< '|' << "Пассажиров" << setfill(' ') << setw(table\_col - 10)

<< '|' << "Место назначения" << setfill(' ') << setw(table\_col - 16)

<< '|' << endl;

} // print\_header

void print\_raw(Route\* ptr, Station\* head\_station, Driver\* head\_driver) // Одна строчка

{

Station\* station\_ptr = findElemStation(ptr->id\_station, head\_station);

Bus\* bus\_ptr = findElemBus(ptr->id\_bus, station\_ptr->busHead); // Проверка соответствия Автобуса с вокзалом TODO

Driver\* driver\_ptr = find\_driver(ptr->id\_driver, head\_driver);

// Строка

cout << '|' << setfill('\_') << setw(table\_col \* 8) << '|' << endl;

cout << '|' << station\_ptr->name << setfill(' ') << setw(table\_col - (station\_ptr->name).size())

<< '|' << bus\_ptr->name << setfill(' ') << setw(table\_col - (bus\_ptr->name).size())

<< '|' << driver\_ptr->name << setfill(' ') << setw(table\_col - (driver\_ptr->name).size())

<< '|' << ptr->route\_number << setfill(' ') << setw(table\_col - (int)log10(ptr->route\_number) - 1)

<< '|' << ptr->time << setfill(' ') << setw(table\_col - ptr->time.size())

<< '|' << ptr->tickets << setfill(' ') << setw(table\_col - (int)log10(ptr->tickets) - 1)

<< '|' << ptr->passengers << setfill(' ') << setw(table\_col - (int)log10(ptr->passengers) - 1)

<< '|' << ptr->end\_route << setfill(' ') << setw(table\_col - ptr->end\_route.size())

<< '|' << endl;

}

//------Перемещение------//

int switch\_move(Station\* s\_head, Station\*& curr\_station, Bus\*& curr\_bus)

{

int loc\_choice = 0;

int step = 0; // Шаг

int id\_station; // id вокзала

if (!s\_head) return 8;

cout << "Выберете список, по которому нужно пройти: \n1. Вокзалов\n2. Автобусов\n>>";

cin >> loc\_choice;

cout << "\nНа сколько нужно шагнуть? (Пример: 2 - на два элемента вперед, -2 - на два элемента назад) \n>>";

cin >> step;

switch (loc\_choice)

{

case 1:

cout << "Текущая позиция: " << curr\_station->id << " " << curr\_station->name << endl;

curr\_station = move\_in\_stations(curr\_station, step);

cout << "Измененная позиция: " << curr\_station->id << " " << curr\_station->name << endl;

break;

case 2:

cout << "Введите id вокзала, по автобусам которого нужно перемещаться: \n>>"; cin >> id\_station;

// Проверка id

if (cin.fail()) return 1;

else if (!findElemStation(id\_station, s\_head)) return 4;

// Если стартовать неоткуда; иначе Если мы не продолжаем движение по тому же списку автобусов, что и в прошлый раз, то стартуем от начала другого выбранного списка

if (!curr\_bus) curr\_bus = findElemStation(id\_station, s\_head)->busHead;

else if (id\_station != curr\_bus->id) curr\_bus = findElemStation(id\_station, s\_head)->busHead;

cout << "Текущая позиция: " << curr\_bus->id\_bus << " " << curr\_bus->name << endl;

curr\_bus = move\_in\_buses(curr\_bus, step);

cout << "Измененная позиция: " << curr\_bus->id\_bus << " " << curr\_bus->name << endl;

break;

default:

cout << "Некорректное значение!" << endl;

break;

} // switch

return 0;

} // switch\_move

//------ОСТАЛЬНЫЕ ФУНКЦИИ------//

void PrintMess(int code)

{

switch (code)

{

case 1:

cout << "Несоответствие типов полей структуры с подаваемым значением!" << endl;

break;

case 2:

cout << "Такой элемент списка уже существует!" << endl;

break;

case 3:

cout << "Все id должны быть уникальными!" << endl;

break;

case 4:

cout << "Элемента нету в этой БД!" << endl;

break;

case 5:

cout << "Номера рейсов не должны повторяться!" << endl;

break;

case 6:

cout << "Названия вокзалов, марок, ФИО водителей должны быть уникальными!" << endl;

break;

case 7:

cout << "Билетов должно быть >= числа пассажиров!" << endl;

break;

case 8:

cout << "База данных все еще не инициализирована! Перед началом работы с БД, выберете опцию 0. Создать БД." << endl;

break;

default:

cout << "Неизвестная ошибка." << endl;

break;

}

}

lists.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

// "перемещение в прямом и обратном направлениях" - ?

// С Т Р У К Т У Р Ы

struct Bus // Двунаправленный кольцевой АВТОБУСЫ

{

Bus\* prev;

Bus\* next;

int id; // ID автовокзала

int id\_bus; // ID автобуса

string name; // Назв марки

};

struct Station // Двунаправленный кольцевой АВТОВОКЗАЛЫ

{

Station\* prev;

Station\* next;

int id; // ID

string name; // Назв автовокзала

Bus\* busHead = NULL; // Голова списка автобусов для данного вокзала

Bus\* busEnd = NULL; // Хвост списка автобусов

};

struct Driver // Водитель (линейный)

{

Driver\* next;

string name; // ФИО

int id; // ID водителя

};

struct Route { // Линейный (рейсы)

Route\* next;

int id\_station; // ID автовокзала

int id\_bus; // ID автобуса

int id\_driver; // ID водителя

int route\_number; // Номер рейса

string time; // Время отправления

int tickets; // Количество проданных билетов

int passengers; // Количество пассажиров

string end\_route; // Место назначения

};

// Функции кольцевого двухсвязного для ВОКЗАЛОВ

int makeStation(int id, string name, Station\*& end, Station\*& head); // Добавление нового элемента

int max\_station\_id(Station\* head); // Найти макс id

void DeleteStation(Station\*& head, Station\*& end, int id); // Удаление элемента по его id

void printStation(Station\* head); // Печать всего списка

void printStationBack(Station\* end); // В обратную сторону

void delListStation(Station\*& head, Station\*& end); // Удаление списка

Station\* findElemStation(int id, Station\* head); // Найти элемент по id

int find\_id\_station(string name, Station\* head); // Найти id по названию

Station\* move\_in\_stations(Station\* curr\_pos, int step); // Перемещение в прямом и обратном направлениях

// Функции кольцевого двухсвязного для АВТОБУСОВ

int makeBus(int id, int id\_bus, string name, Bus\*& end, Bus\*& head); // Добавление нового элемента

int max\_bus\_id(Bus\* head); // Найти макс id

void DeleteBus(Bus\*& head, Bus\*& end, int index); // Удаление элемента по его индексу

void printBus(Bus\* head, Station\* s\_head); // Печать всего списка

void printBusBack(Bus\* end); // В обратную сторону

void delListBus(Bus\*& head, Bus\*& end); // Удаление списка

Bus\* findElemBus(int id, Bus\* head); // Найти элемент в списке по id

int find\_id\_bus(string name, Bus\* head); // Найти id по названию

Bus\* move\_in\_buses(Bus\* curr\_pos, int step); // Перемещение в прямом и обратном направлениях

void print\_all\_buses(Station\* head); // Печать всех автобусов

// Функции линейного ВОДИТЕЛЬ

int make\_driver(int id, string name, Driver\*& end, Driver\*& head); // Добавление нового элемента

int max\_driver\_id(Driver\* head); // Найти макс id

void print\_drivers(Driver\* head); // Печать всего списка

Driver\* find\_driver(int id, Driver\* head); // Найти элемент в списке по id

int find\_id\_driver(string name, Driver\* head); // Найти id по названию

void delete\_driver(Driver\*& head, Driver\*& end, int id); // Удаление элемента по его id

void del\_drivers(Driver\*& head); // Удаление списка

// Функции линейного РЕЙСЫ

int make\_route(Route\* ptr, Route\*& end, Route\*& head); // Добавление нового элемента

void print\_routes(Route\* head, Station\* s\_head, Driver\* d\_head); // Печать всего списка

Route\* find\_route\_stationID(int id\_station, Route\* head); // Найти элемент в списке по полю id вокзала

Route\* find\_route\_busID(int id\_bus, Route\* head); // Найти элемент в списке по полю id автобуса

Route\* find\_route\_driverID(int id\_driver, Route\* head); // Найти элемент в списке по полю id водителя

Route\* find\_route\_number(int route\_number, Route\* head); // Найти элемент в списке по полю номера маршрута

Route\* find\_route\_time\_dep(string time\_dep, Route\* head);

Route\* find\_route\_tickets(int tickets, Route\* head);

Route\* find\_route\_pass(int passengers, Route\* head);

Route\* find\_route\_end(string end\_route, Route\* head);

void print\_stations\_in\_route(Route\* r\_head, Station\* s\_head); // Вывод всех вокзалов для удаления или поиска

void print\_buses\_in\_route(Route\* r\_head, Station\* s\_head); // Вывод всех вокзалов и автобусов для удаления или поиска

void print\_drivers\_in\_route(Route\* r\_head, Driver\* d\_head); // Вывод всех водителей для удаления или поиска

void delete\_route(Route\*& head, Route\*& end, int route\_number); // Удаление элемента по номеру маршрута

void del\_routes(Route\*& head); // Удаление списка

lists.cpp

#include <iostream>

#include <string>

#include "lists.h"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Р Е А Л И З А Ц И Я Ф У Н К Ц И Й \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Функции кольцевого двухсвязного для марок

int makeBus(int id, int id\_bus, string name, Bus\*& end, Bus\*& head) // Добавление нового элемента

{

if (find\_id\_bus(name, head) != -1) return 6;

Bus\* ptr = new Bus;

if (!head) head = ptr;

else

{

end->next = ptr;

ptr->prev = end;

}

// Заполнение инф полей

ptr->id = id;

ptr -> id\_bus = id\_bus;

ptr->name = name;

end = ptr;

end->next = head;

head->prev = end;

return 0;

} // makeBus

int max\_bus\_id(Bus\* head) // Найти макс id

{

Bus\* ptr = head;

int max\_id = 0;

do {

if (max\_id < ptr->id\_bus) max\_id = ptr->id\_bus;

ptr = ptr->next;

} while (ptr != head);

return max\_id;

}

void DeleteBus(Bus\*& head, Bus\*& end, int id) // Удаление элемента по его id

{ // TODO: проверка индекса

int counter = 0;

Bus\* ptr = head;

do

{

if (id == ptr->id\_bus)

{

if (counter == 0) // Удаляется голова

{

ptr->prev->next = ptr->next; // Лево на право

ptr->next->prev = ptr->prev; // Право на лево

head = ptr->next; // Голову на второй

delete ptr;

return;

}// if

else // Из промежутка или конца

{

ptr->prev->next = ptr->next; // Лево на право

ptr->next->prev = ptr->prev; // Право на лево

delete ptr;

return;

} // if

}

counter++;

ptr = ptr->next;

} while (ptr != head);

} // DeleteBus

void printBus(Bus\* head, Station\* s\_head) // Печать всего списка

{

Bus\* ptr = head;

if (ptr) cout <<" "<<findElemStation(ptr->id, s\_head)->name << " " << ptr->name << endl;

else

{

//cout << "Список пуст!" << endl;

return;

} // else

ptr = ptr->next;

while (ptr != head)

{

cout << " " << findElemStation(ptr->id, s\_head)->name << " " << ptr->name << endl;

ptr = ptr->next;

} // while

} // printBus

void printBusBack(Bus\* end) // В обратную сторону

{

Bus\* ptr = end;

if (ptr) cout << ptr->id << " " << ptr->name << endl;

else

{

cout << "Список пуст!" << endl;

return;

} // else

ptr = ptr->prev;

while (ptr != end)

{

cout << ptr->id << " " << ptr->name << endl;

ptr = ptr->prev;

} // while

} // printBusBack

void delListBus(Bus\*& head, Bus\*& end) // Удаление списка

{

Bus\* ptr = head;

while (head != end)

{

ptr = head;

ptr->prev->next = ptr->next; // Лево на право

ptr->next->prev = ptr->prev; // Право на лево

head = ptr->next;

delete ptr;

}

delete head; // Удаляем конец

head = NULL; end = NULL;

} // delListBus

Bus\* findElemBus(int id, Bus\* head) // Найти элемент по id

{

Bus\* ptr = head;

if (!ptr) return NULL; // Если список пуст - выходим

do // Поиск по полю id

{

if (ptr->id\_bus == id) return ptr;

ptr = ptr->next;

} while (ptr != head);

return NULL;

} // findElemStation

int find\_id\_bus(string name, Bus\* head) // Найти id по названию

{

Bus\* ptr = head;

if (!ptr) return -1; // Если список пуст - выходим

do // Поиск по полю id

{

if (ptr->name == name) return ptr->id\_bus;

ptr = ptr->next;

} while (ptr != head);

return -1;

}

Bus\* move\_in\_buses(Bus\* curr\_pos, int step) // Перемещение в прямом и обратном направлениях

{

if (step >= 0) // Вперед

{

for (int counter = 0; counter < step; counter++)

curr\_pos = curr\_pos->next;

}

else // Шаг отрицательный => назад!

{

for (int counter = 0; counter < step \* (-1); counter++)

curr\_pos = curr\_pos->prev;

}

return curr\_pos;

}

void print\_all\_buses(Station\* head)

{

cout << "ВСЕ АВТОБУСЫ В БАЗЕ ДАННЫХ" << endl;

Station\* ptr = head;

do

{

printBus(ptr->busHead, head);

ptr = ptr->next;

} while (ptr != head);

}

//Bus\* findElemBus(float a, Bus\* head); // Найти элемент в списке

// Функции двухсвязного списка для вокзалов

int makeStation(int id, string name, Station\*& end, Station\*& head) // Добавление нового элемента

{

if (find\_id\_station(name, head) != -1) return 6; // Повтор вокзала

else if (findElemStation(id, head)) return 3; // Повтор id

Station\* ptr = new Station;

if (!head) head = ptr;

else

{

end->next = ptr;

ptr->prev = end;

}

// Заполнение инф полей

ptr->id = id;

ptr->name = name;

end = ptr;

end->next = head;

head->prev = end;

return 0;

}

int max\_station\_id(Station\* head) // Найти макс id

{

Station\* ptr = head;

int max\_id = 0;

do {

if (max\_id < ptr->id) max\_id = ptr->id;

ptr = ptr->next;

} while (ptr != head);

return max\_id;

}

void DeleteStation(Station\*& head, Station\*& end, int id) // Удаление элемента по его id

{

int counter = 0;

Station\* ptr = head;

do

{

if (id == ptr->id)

{

if (counter == 0) // Удаляется голова

{

ptr->prev->next = ptr->next; // Лево на право

ptr->next->prev = ptr->prev; // Право на лево

head = ptr->next; // Голову на второй

delete ptr;

return;

}// if

else // Из промежутка или конца

{

ptr->prev->next = ptr->next; // Лево на право

ptr->next->prev = ptr->prev; // Право на лево

delete ptr;

return;

} // if

}

counter++;

ptr = ptr->next;

} while (ptr != head);

} // DeleteStation

void printStation(Station\* head) // Печать всего списка

{

Station\* ptr = head;

if (!ptr) {

cout << "Список вокзалов пуст!" << endl;

return;

}

cout << "СПИСОК ВОКЗАЛОВ" << endl;

cout << " " << ptr->name << endl;

ptr = ptr->next;

while (ptr != head)

{

cout <<" " << ptr->name << endl;

ptr = ptr->next;

} // while

}

void printStationBack(Station\* end) // Печать всего списка в обратном порядке

{

Station\* ptr = end;

if (ptr) cout << ptr->id << " " << ptr->name << endl;

else

{

cout << "Список пуст!" << endl;

return;

} // else

ptr = ptr->prev;

while (ptr != end)

{

cout << ptr->id << " " << ptr->name << endl;

ptr = ptr->prev;

} // while

}

void delListStation(Station\*& head, Station\*& end) // Удаление списка

{

Station\* ptr = head;

while (head != end)

{

ptr = head;

ptr->prev->next = ptr->next; // Лево на право

ptr->next->prev = ptr->prev; // Право на лево

head = ptr->next;

delete ptr;

}

delete head; // Удаляем конец

head = NULL; end = NULL;

} // delListRDL

Station\* findElemStation(int id, Station\* head) // Найти элемент по id

{

if (!head) return NULL;

Station\* ptr = head;

do // Поиск по полю id

{

if (ptr->id == id) return ptr;

ptr = ptr->next;

} while (ptr != head);

return NULL;

} // findElemStation

int find\_id\_station(string name, Station\* head)

{

if (!head) return -1;

Station\* ptr = head;

do // Поиск по полю id

{

if (ptr->name == name) return ptr->id;

ptr = ptr->next;

} while (ptr != head);

return -1;

} // find\_id\_station

Station\* move\_in\_stations(Station\* curr\_pos, int step) // Перемещение в прямом и обратном направлениях

{

if(step>=0) // Вперед

{

for (int counter = 0; counter < step; counter++)

curr\_pos = curr\_pos->next;

}

else // Шаг отрицательный => назад!

for (int counter = 0; counter < step\*(-1); counter++)

curr\_pos = curr\_pos->prev;

return curr\_pos;

}

// Функции линейного ВОДИТЕЛЬ

int make\_driver(int id, string name, Driver\*& end, Driver\*& head) // Добавление нового элемента

{

if (find\_driver(id, head)) return 3;

else if (find\_id\_driver(name, head)) return 6;

Driver\* ptr = new Driver; // Новый элемент

if (!head) head = ptr; // Если это первый элемент

else { end->next = ptr; } // У конца появилось продолжение

// Заполнение информ полей

ptr->id = id;

ptr->name = name;

end = ptr; // Делаем этот элемент последним

ptr->next = NULL; // Следующих элементов списка нет

return 0;

} // make

int max\_driver\_id(Driver\* head) // Найти макс id

{

Driver\* ptr = head;

int max\_id = 0;

while(ptr)

{

if (max\_id < ptr->id) max\_id = ptr->id;

ptr = ptr->next;

}

return max\_id;

}

void print\_drivers(Driver\* head) // Печать всего списка

{

cout << "СПИСОК ВОДИТЕЛЕЙ" << endl;

Driver\* ptr = head;

if (!ptr) cout << "Список пуст" << endl;

while (ptr)

{

cout << " " << ptr->name << endl; // Считывание данного

ptr = ptr->next; // Переход к следующему

}

cout << endl;

} // print

Driver\* find\_driver(int id, Driver\* head) // Найти элемент по id

{

Driver\* ptr = head;

while (ptr)

{

if (ptr->id == id) return ptr;

ptr = ptr->next;

}

return NULL;

} // findElemStation

int find\_id\_driver(string name, Driver\* head) // Найти id по названию

{

Driver\* ptr = head;

while (ptr)

{

if (ptr->name == name) return ptr->id;

ptr = ptr->next;

}

return NULL;

}

void delete\_driver(Driver\*& head, Driver\*& end, int id) { // Удаление элемента по id

Driver\* ptr = head;

if (ptr->id == id) // Удаление головы

{

head = ptr->next;

delete ptr;

return;

} // if

while(ptr->next->id != id) ptr = ptr->next; // Ищем предшественника удаляемого элемента

Driver\* ptr2 = ptr->next;

ptr->next = ptr->next->next;

if (ptr2 == end) end = ptr;

delete ptr2;

} // delete\_driver

void del\_drivers(Driver\*& head) // Удаление всего списка

{

Driver\* ptr;

while (head) // Удаление, начинающееся с "головы"

{

ptr = head->next;

delete head;

head = ptr;

} // while

head = NULL;

} // delList

// Функции линейного РЕЙСЫ

int make\_route(Route\* ptr, Route\*& end, Route\*& head) // Добавление нового элемента

{

if (find\_route\_number(ptr->route\_number, head)) return 5;

if (!head) head = ptr; // Если это первый элемент

else { end->next = ptr; } // У конца появилось продолжение

end = ptr; // Делаем этот элемент последним

ptr->next = NULL; // Следующих элементов списка нет

return 0;

} // make

// ПОИСК ПО РАЗНЫМ ПОЛЯМ

Route\* find\_route\_stationID(int id\_station, Route\* head) // Найти элемент в списке по полю id вокзала

{

Route\* ptr = head;

while (ptr)

{

if (ptr->id\_station == id\_station) return ptr;

ptr = ptr->next;

}

return NULL;

}

Route\* find\_route\_busID(int id\_bus, Route\* head) // Найти элемент в списке по полю id автобуса

{

Route\* ptr = head;

if (id\_bus == -1) return NULL;

while (ptr)

{

if (ptr->id\_bus == id\_bus) return ptr;

ptr = ptr->next;

}

return NULL;

}

Route\* find\_route\_driverID(int id\_driver, Route\* head) // Найти элемент в списке по полю id водителя

{

Route\* ptr = head;

while (ptr)

{

if (ptr->id\_driver == id\_driver) return ptr;

ptr = ptr->next;

}

return NULL;

}

Route\* find\_route\_number(int route\_number, Route\* head) // Найти элемент по номеру маршрута

{

Route\* ptr = head;

while (ptr)

{

if (ptr->route\_number == route\_number) return ptr;

ptr = ptr->next;

}

return NULL;

} // find\_route\_number

Route\* find\_route\_time\_dep(string time\_dep, Route\* head) // По времени

{

Route\* ptr = head;

while (ptr)

{

if (ptr->time == time\_dep) return ptr;

ptr = ptr->next;

}

return NULL;

}

Route\* find\_route\_tickets(int tickets, Route\* head) // По билетам

{

Route\* ptr = head;

while (ptr)

{

if (ptr->tickets == tickets) return ptr;

ptr = ptr->next;

}

return NULL;

}

Route\* find\_route\_pass(int passengers, Route\* head) // По пассажирам

{

Route\* ptr = head;

while (ptr)

{

if (ptr->passengers == passengers) return ptr;

ptr = ptr->next;

}

return NULL;

}

Route\* find\_route\_end(string end\_route, Route\* head) // По пункту

{

Route\* ptr = head;

while (ptr)

{

if (ptr->end\_route == end\_route) return ptr;

ptr = ptr->next;

}

return NULL;

}

// ПЕЧАТЬ

void print\_routes(Route\* head, Station\* s\_head, Driver\* d\_head) // Печать всего списка

{

Route\* ptr = head;

Station\* s\_ptr; // Проход по вокзалам

if (!ptr) cout << "Список пуст" << endl;

while (ptr)

{

s\_ptr = findElemStation(ptr->id\_station, s\_head);

cout << "\t" << s\_ptr->name << " " << findElemBus(ptr->id\_bus, s\_ptr->busHead)->name << " "

<< find\_driver(ptr->id\_driver, d\_head)->name << " " << ptr->route\_number << " " << ptr->time << " "

<< ptr->tickets << " " << ptr->passengers << " " << ptr->end\_route << endl; // Считывание данного

ptr = ptr->next; // Переход к следующему

}

cout << endl;

} // print

// УДАЛЕНИЕ

void delete\_route(Route\*& head, Route\*& end, int route\_number) { // Удаление элемента по номеру маршрута

Route\* ptr = head;

if (ptr->route\_number == route\_number) // Удаление головы

{

head = ptr->next;

delete ptr;

return;

} // if

while (ptr->next->route\_number != route\_number) ptr = ptr->next; // Ищем предшественника удаляемого элемента

Route\* ptr2 = ptr->next;

ptr->next = ptr->next->next;

if (ptr2 == end) end = ptr;

delete ptr2;

} // Delete

void del\_routes(Route\*& head) // Удаление всего списка

{

Route\* ptr;

while (head) // Удаление, начинающееся с "головы"

{

ptr = head->next;

delete head;

head = ptr;

} // while

head = NULL;

} // delList

void print\_stations\_in\_route(Route\* r\_head, Station\* s\_head)

{

Station\* s\_ptr = s\_head;

cout << "Все вокзалы с рейсами: " << endl;

do

{

if (find\_route\_stationID(s\_ptr->id, r\_head))

cout << s\_ptr->id << " " << s\_ptr->name << endl;

s\_ptr = s\_ptr->next;

} while (s\_ptr != s\_head);

}

void print\_buses\_in\_route(Route\* r\_head, Station\* s\_head)

{

Route\* r\_ptr = r\_head;

Station\* temp;

cout << "Все вокзалы и автобусы с рейсами: " << endl;

while (r\_ptr)

{

temp = findElemStation(r\_ptr->id\_station, s\_head);

cout << r\_ptr->id\_station << " (" << temp->name << ") "

<< r\_ptr->id\_bus << " (" << findElemBus(r\_ptr->id\_bus, temp->busHead)->name << ")" << endl;

r\_ptr = r\_ptr->next;

}

}

void print\_drivers\_in\_route(Route\* r\_head, Driver\* d\_head)

{

Driver\* d\_ptr = d\_head;

cout << "Все водители с рейсами: " << endl;

while (d\_ptr)

{

if (find\_route\_driverID(d\_ptr->id, r\_head))

cout << d\_ptr->id << " " << d\_ptr->name << endl;

d\_ptr = d\_ptr->next;

}

}

# 

# Тестирование программы

## Тестирование некорректных режимов

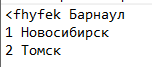
Тестирование функций записи из файлов

Тест 1

**Цель**: Проверить работу программы при строке в поле id

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

**Ожидаемый результат**: «Несоответствие типов полей структуры с подаваемым значением!»

**Полученный результат**:

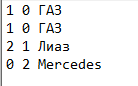


Тест 2

**Цель**: Проверить работу программы при наличии одинаковых данных в одном файле

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

**Ожидаемый результат**:

«Все id должны быть уникальными!»

**Полученный результат**:



Тест 3

**Цель**: Проверить работу программы при повторяющихся номерах рейсов

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

**Ожидаемый результат**:

«Номера рейсов не должны повторяться!»

**Полученный результат**:



Тест 4

**Цель**: Проверить работу программы при одинаковых ФИО водителей в файле

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

**Ожидаемый результат**:

«Названия вокзалов, марок, ФИО водителей должны быть уникальными!»

**Полученный результат**:



Тестирование функций добавления элементов

Тест 5

**Цель**: Проверить работу программы при попытке добавить автобус в несуществующий вокзал.

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

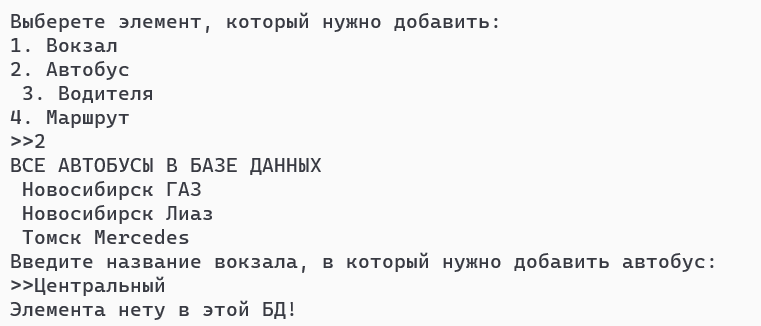
 

Несуществующий вокзал: Центральный

**Ожидаемый результат**:

«Элемента нету в этой БД!»

**Полученный результат**:



Тест 6

**Цель**: Проверить работу программы при попытке добавить рейс с несуществующим автобусом.

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

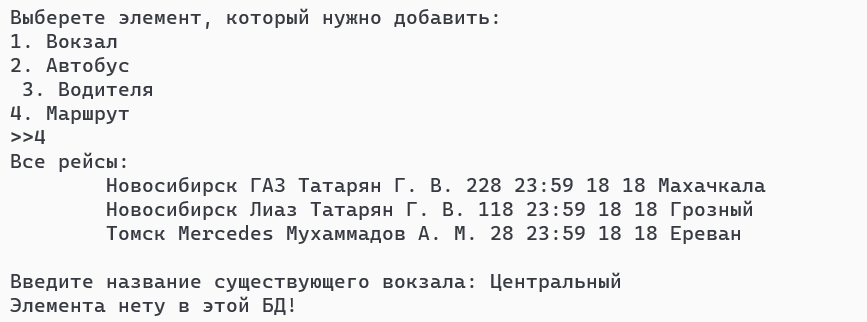
 

Несуществующий автобус:

**Ожидаемый результат**:

«Элемента нету в этой БД!»

**Полученный результат**:



Тестирование функций удаления элементов

Тест 7

**Цель**: Проверить работу программы при попытке удалить несуществующий вокзал.

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

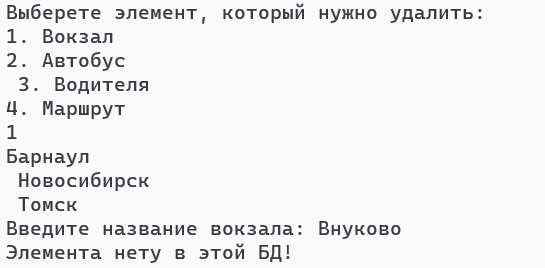
 

Несуществующий вокзал - Внуково

**Ожидаемый результат**:

«Элемента нету в этой БД!»

**Полученный результат**:



Тест 8

**Цель**: Проверить работу программы при попытке удалить несуществующий автобус

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

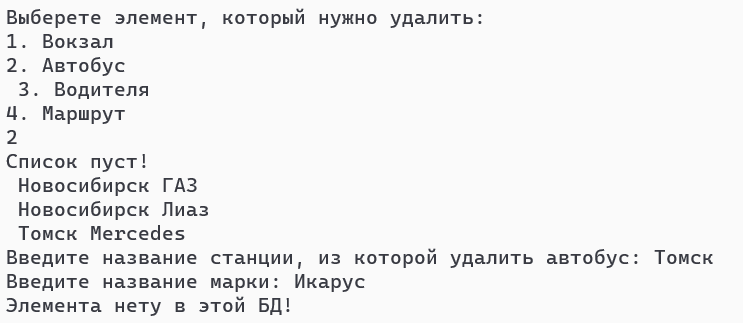
 

Несуществующий автобус - Икарус

**Ожидаемый результат**:

«Элемента нету в этой БД!»

**Полученный результат**:



Тестирование функций поиска элементов

Тест 9

**Цель**: Проверить работу программы при попытке найти несуществующий автобус

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

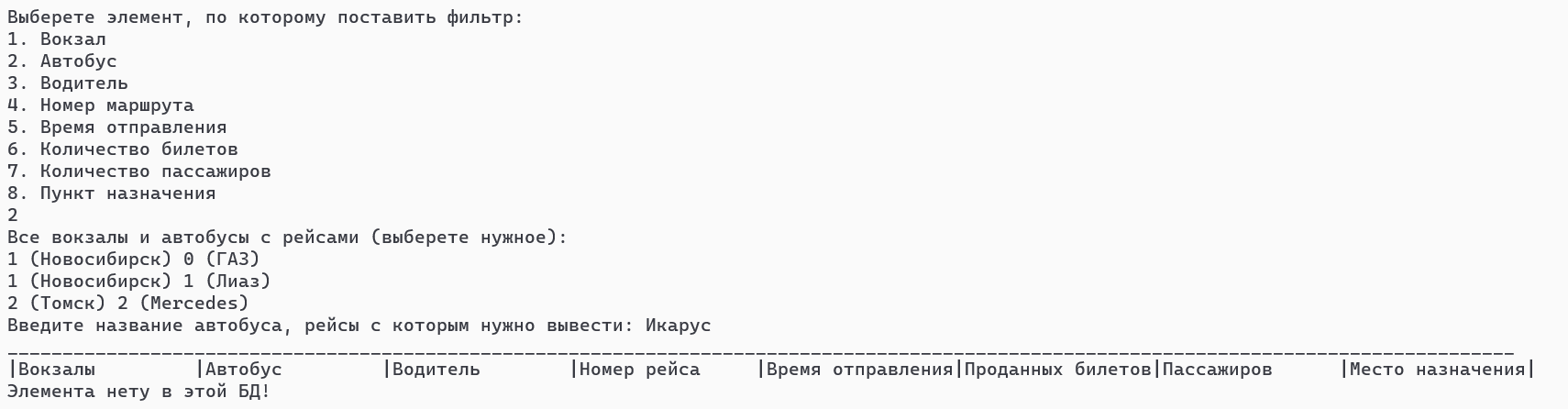
 

Несуществующий автобус - Икарус

**Ожидаемый результат**:

«Элемента нету в этой БД!»

**Полученный результат**:



Тест 10

**Цель**: Проверить работу программы при попытке найти рейс с неправильно заданным временем

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

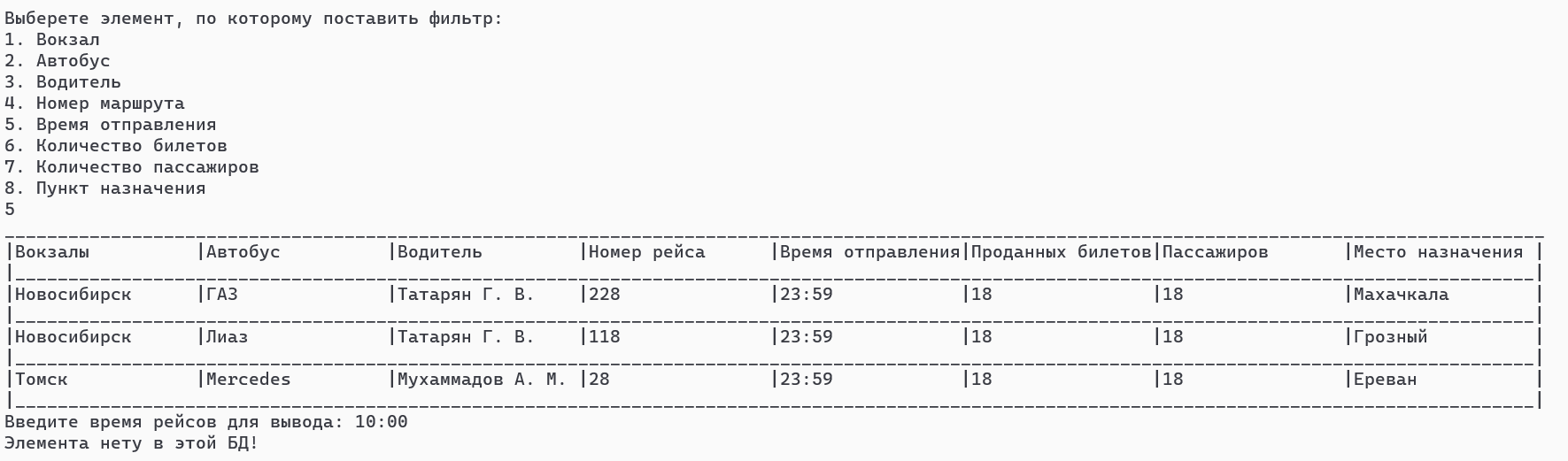
 

Запрашиваемое время – 10:00

**Ожидаемый результат**:

«Элемента нету в этой БД!»

**Полученный результат**:



## Тестирование корректных режимов

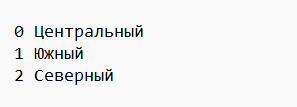
Тестирование функции вывода таблицы рейсов.

Тест 1

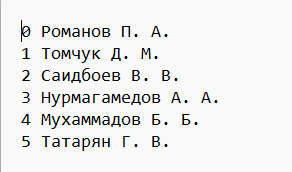
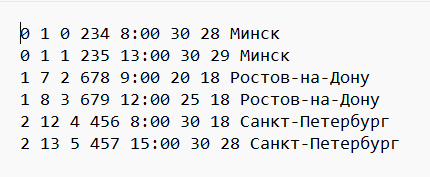
**Цель**: Проверить функцию вывода таблицы при корректных входных данных

**Исходные данные**:

Вокзалы Марки

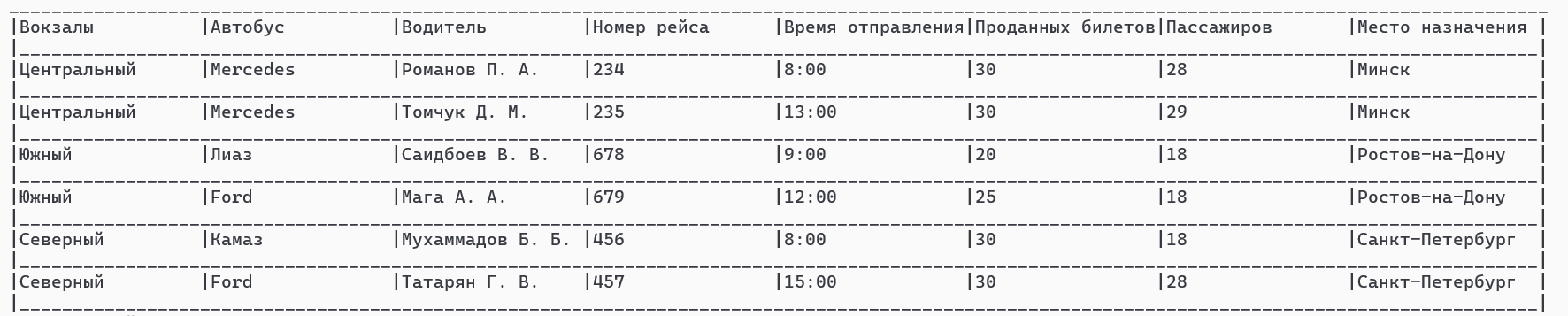
Водители Рейсы

**Ожидаемый результат**:

Таблица рейсов с id, замененными на соответствующие названия.

**Полученный результат**:



Проверка функции добавления

Тест 2

**Цель**: Проверить работу программы при попытке вставить нового водителя.

**Исходные данные**: Те же самые, водитель – Бустиренко Г. В.

**Ожидаемый результат**:

Обновленный список водителей:

Романов П. А.

Томчук Д. М.

Саидбоев В. В.

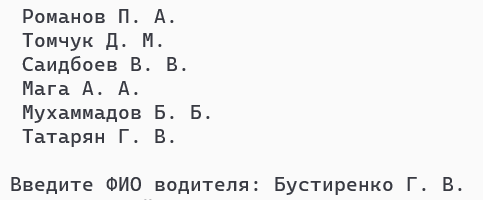
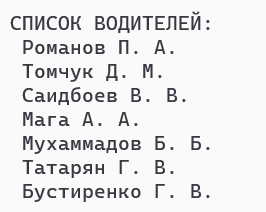
Мага А. А.

Мухаммадов Б. Б.

Татарян Г. В.

Бустиренко Г. В.

**Полученный результат**:

Тест 3

**Цель**: Создать новый рейс с добавленным водителем

**Исходные данные**:

Те же; поля нового рейса:

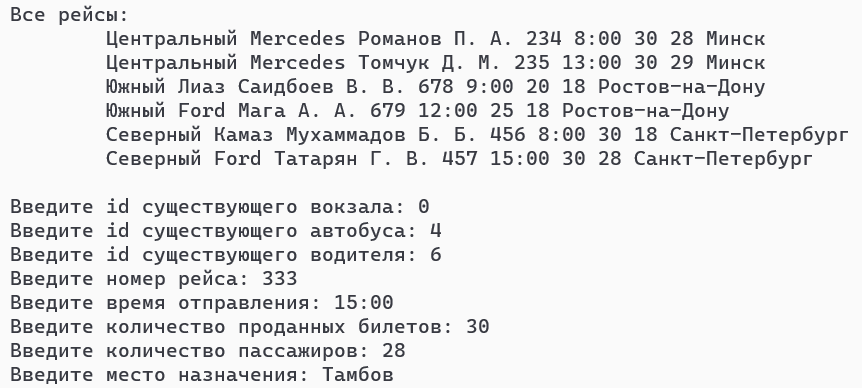
0 4 6 333 15:00 30 28 Тамбов

**Ожидаемый результат**:

Новая строка таблицы:

Центральный Камаз Бустиренко Г. В. 333 15:00 30 28 Тамбов

**Полученный результат**:





Тестирование функций удаления

Тест 4

**Цель**: Проверить работу программы при запросе удалить автобус, не находящийся ни в одном из рейсов, из существующего вокзала

**Исходные данные**:

Те же, что в тесте 1.

Вокзал: Центральный;

Удаляемый автобус: Камаз

**Ожидаемый результат**:

Обновленный список всех автобусов:

Центральный Mercedes

Центральный Hyondai

Центральный Лиаз

Южный Mercedes

Южный Hyondai

Южный Лиаз

Южный Ford

Северный Mercedes

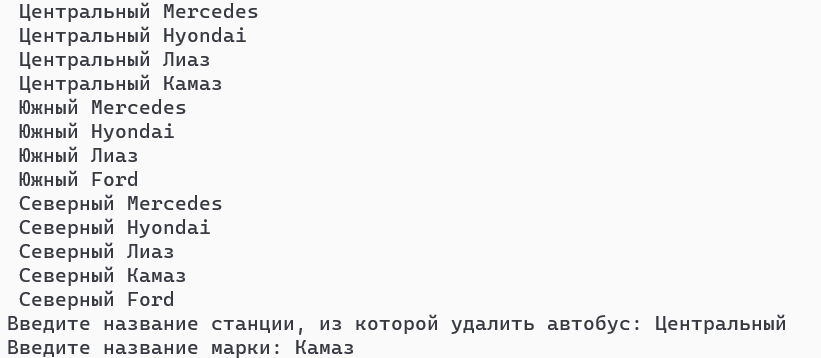
Северный Hyondai

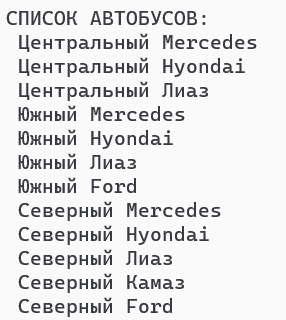
Северный Лиаз

Северный Камаз

Северный Ford

**Полученный результат**:





Тест 5

**Цель**: Проверить работу программы при запросе удалить автобус, находящийся в одном из рейсов, из существующего вокзала

**Исходные данные**:

Те же, что в тесте 1.

Вокзал: Центральный;

Удаляемый автобус: Mercedes

**Ожидаемый результат**:

Обновленный список всех автобусов:

Центральный Hyondai

Центральный Лиаз

Южный Mercedes

Южный Hyondai

Южный Лиаз

Южный Ford

Северный Mercedes

Северный Hyondai

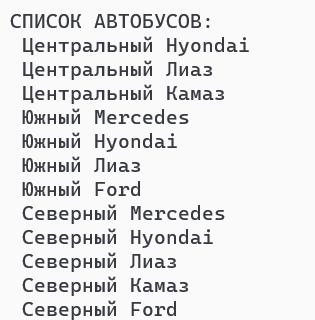
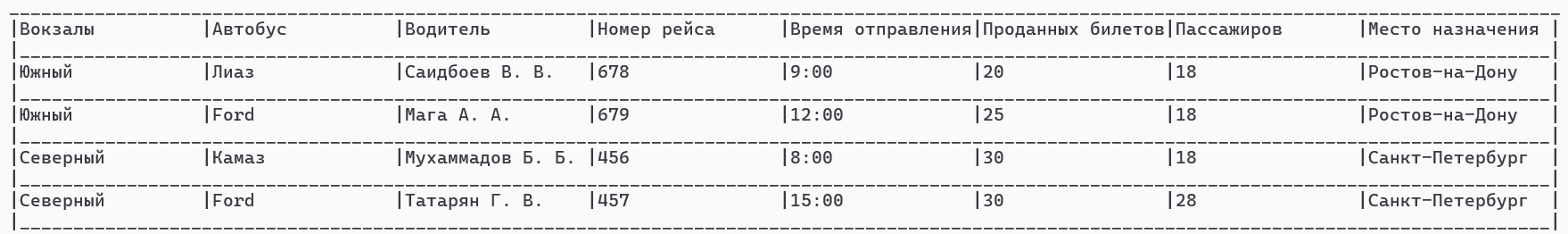
Северный Лиаз

Северный Камаз

Северный Ford

А также таблица рейсов без строк с удаленным автобусом.

**Полученный результат**:

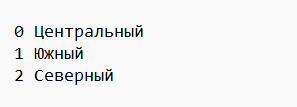
Тестирование функций поиска

Тест 6

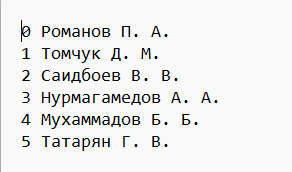
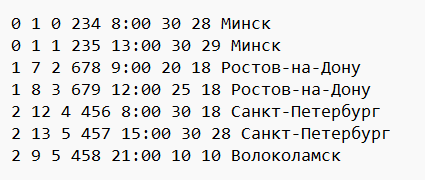
**Цель**: Проверить работу программы при поиске всех рейсов с заданным названием автобуса

**Исходные данные**:

Вокзалы Марки

Водители Рейсы

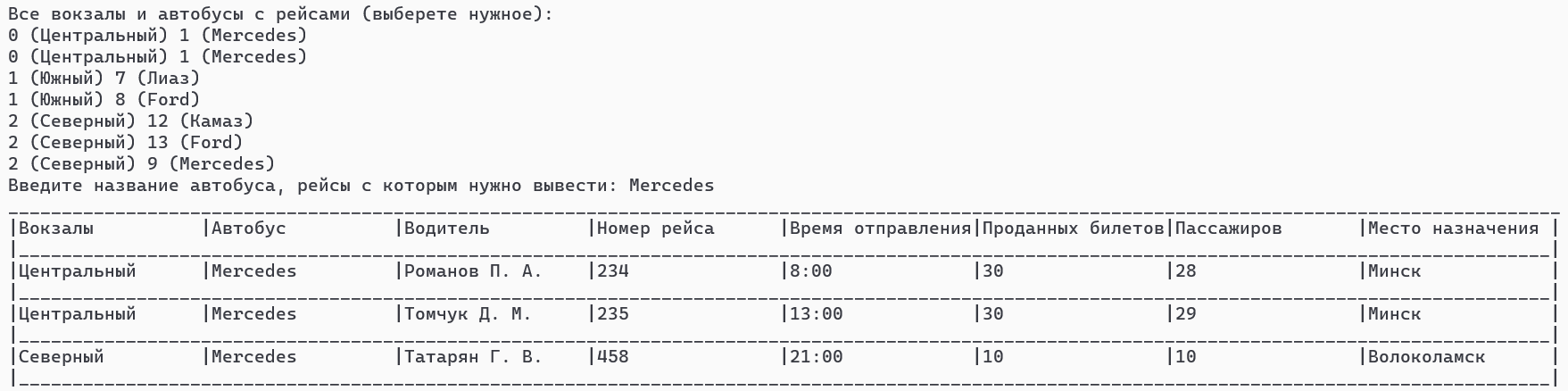
 

Автобус: Mercedes

**Ожидаемый результат:**

Таблица с автобусами Mercedes

**Полученный результат**:



Тест 7

**Цель**: Проверить работу программы при поиске всех рейсов с заданным временем отправления

**Исходные данные**:

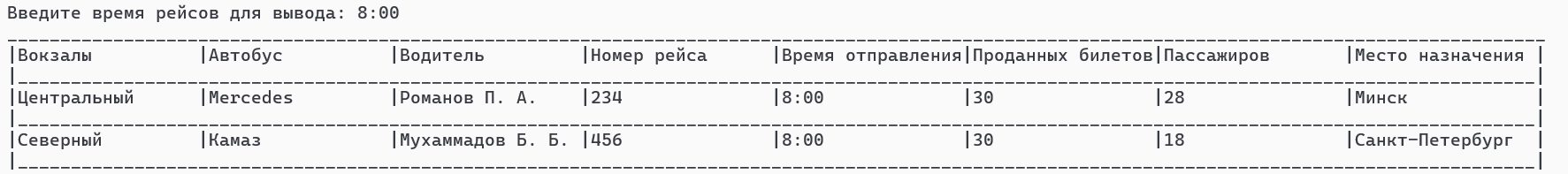
Те же, что в тесте 6

Время – 8:00

**Ожидаемый результат:**

Таблица рейсов с временем отправления 8:00

**Полученный результат**:



Тестирование функций печати

Тест 8

**Цель**: Проверить работу программы при запросе печати всех списков

**Исходные данные**:

Те же, что в тесте 6

**Ожидаемый результат:**

СПИСОК ВОКЗАЛОВ:

Центральный

Южный

Северный

СПИСОК АВТОБУСОВ:

Центральный Mercedes

Центральный Hyondai

Центральный Лиаз

Центральный Камаз

Южный Mercedes

Южный Hyondai

Южный Лиаз

Южный Ford

Северный Mercedes

Северный Hyondai

Северный Лиаз

Северный Камаз

Северный Ford

СПИСОК ВОДИТЕЛЕЙ:

Романов П. А.

Томчук Д. М.

Саидбоев В. В.

Мага А. А.

Мухаммадов Б. Б.

Татарян Г. В.

СПИСОК МАРШРУТОВ:

Центральный Mercedes Романов П. А. 234 8:00 30 28 Минск

Центральный Mercedes Томчук Д. М. 235 13:00 30 29 Минск

Южный Лиаз Саидбоев В. В. 678 9:00 20 18 Ростов-на-Дону

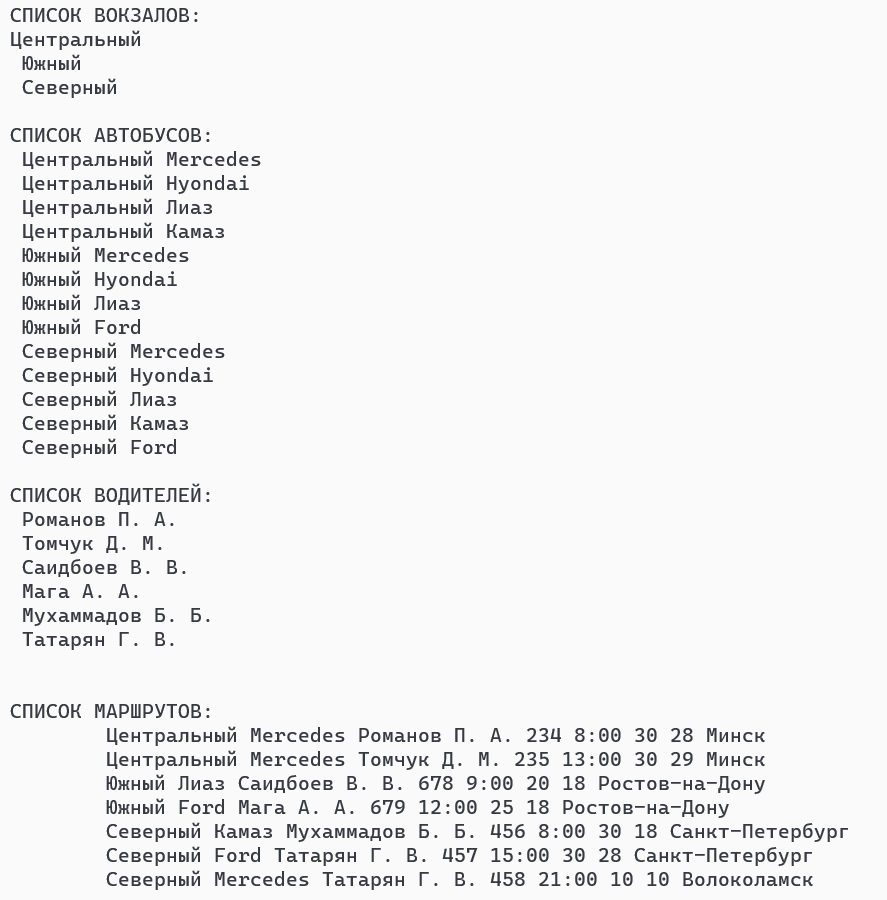
Южный Ford Мага А. А. 679 12:00 25 18 Ростов-на-Дону

Северный Камаз Мухаммадов Б. Б. 456 8:00 30 18 Санкт-Петербург

Северный Ford Татарян Г. В. 457 15:00 30 28 Санкт-Петербург

Северный Mercedes Татарян Г. В. 458 21:00 10 10 Волоколамск

**Полученный результат**:



Тестирование функции прохода по спискам

Тест 9

**Цель**: Проверить работу программы при запросе совершить проход в положительном направлении по кольцевому списку автобусов внутри Центрального вокзала

**Исходные данные**:

Те же, что в тесте 6

Id вокзала = 0

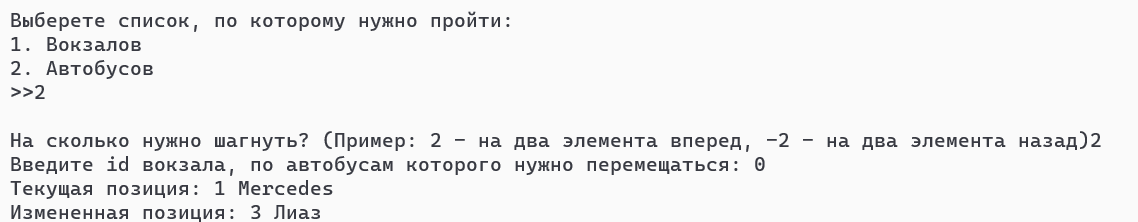
Шагов прохода: 2

**Ожидаемый результат:**

Текущая позиция: 1 Mercedes

Измененная позиция: 3 Лиаз

**Полученный результат:**

****

Тест 10

**Цель**: Проверить работу программы при запросе совершить проход в отрицательном направлении по кольцевому списку автобусов внутри Центрального вокзала, вернувшись тем самым в изначальную позицию в тесте 9.

**Исходные данные**:

Те же, что в тесте 6

Id вокзала = 0

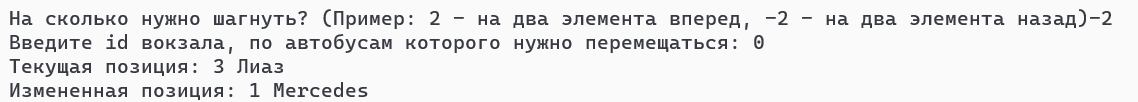
Шагов прохода: -2

**Ожидаемый результат:**

Текущая позиция: 3 Лиаз

Измененная позиция: 1 Mercedes

**Полученный результат:**

****

# Вывод

Разработка программы остановлена на том основании, что ожидаемые результаты тестов совпали с полученными.