МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Кафедра компьютерных технологий и программной инженерии

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. техн. наук |  |  |  | А. В. Туманова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| **РАЗРАБОТКА ПРОГРАММЫ**  **«Маршруты»** |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛА

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТКА ГР. № | 4131 |  | 24.12.2022 |  | Е. А. Кресик |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

СОДЕРЖАНИЕ

[1. Задание на курсовой проект 3](#_Toc122812847)

[2. Описание структур данных 3](#_Toc122812848)

[3. Описание программы и функций 4](#_Toc122812849)

[4. Описание пользовательского интерфейса 9](#_Toc122812857)

[5. Результаты тестирования программы 10](#_Toc122812858)

[5.1 Словесное описание 10](#_Toc122812859)

[5.2 Тестовые данные 12](#_Toc122812879)

[5.3 Скриншоты выполнения тестовых данных 13](#_Toc122812880)

[ЗАКЛЮЧЕНИЕ 27](#_Toc122812881)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc122812882)

[Приложение А. Текст программы 29](#_Toc122812883)

1. Задание на курсовой проект

Задачей курсового проекта является разработка программы для работы с маршрутами транспорта, которая позволяет вводить информацию, хранить её в файле, осуществлять поиск, модификацию, сортировку и удаление данных.

1. Описание структур данных

Данные о поезде хранятся в структуре MARSH.

**struct** MARSH

{

string startinPoint;

string destination;

string routeNumber;

};

Начальный пункт – строка, хранящая название пункта отправления. Может содержать только символы от а до я (в любом регистре, буква ‘ё’ будет заменена на ‘е’), дефис (при условии, что слева и справа от дефиса нет пробелов); длина каждого слова (разделителем между словами является пробел) не меньше 3 символов.

Конечный пункт – строка, хранящая название конечного пункта маршрута. Может содержать только символы от а до я (в любом регистре, буква ‘ё’ будет заменена на ‘е’), дефис (при условии, что слева и справа от дефиса нет пробелов); длина каждого слова (разделителем между словами является пробел) не меньше 3 символов.

Номер маршрута – строка, хранящая номер маршрута. Может содержать только цифры от 0 до 9; максимальное число 9999; число не может начинаться с 0; в числе не может быть пробелов, тире, знаков препинания.

Номер маршрута является уникальным для каждой записи; у двух маршрутов не могут совпадать начальный и конечный пункты соответственно (независимо от регистра символов); у одного маршрута не могут совпадать начальный и конечный пункты (независимо от регистра символов).

База данных хранится в формате .txt.

Формат хранения: Начальный пункт|Конечный пункт|Номер маршрута

Для работы с БД используется односвязный список.

1. Описание программы и функций

Программа реализована на языке C++ в виде консольного приложения. В отдельной функции реализовано меню пользователя, в котором каждому действию соответствует определенная цифра. Реализованы следующие функции для работы с данными:

* 1. Загрузка данных из файла
  2. Добавление маршрута
  3. Удаление маршрута
  4. Вывод списка на экран
  5. Поиск
  6. Сортировка
  7. Редактирование
  8. Удаление всего списка
  9. Сохранение данных в файл

10. Выход

Переходы между пунктами определяются в меню. Реализация всех функций находится в [приложении A.](#ПриложениеА)

# Загрузка данных из файла

**bool** readingFile(List \*\*startList, List \*\*startErrorList);

Производится выбор: добавить элементы из БД в существующий список маршрутов или обновить его.

При чтении файла данные считываются построчно, проверяется наличие в строке двух разделительных знаков (‘|’), в случае корректности строки – курсор перемещается на начало текущей строки и данные считываются последовательно по полям, в случае некорректности строки – она записывается в линейный список ошибочных строк (в конец).

Производится проверка корректности каждого поля и очистка каждого поля от «лишних» пробелов. Производится проверка уникальности маршрута. Если одна из проверок не пройдена – строка добавляется в линейный список ошибочных строк (в конец). Если все проверки пройдены – маршрут добавляется в линейный список корректных маршрутов (в конец).

После выполнения выводится сообщение об успешной загрузке, список корректных маршрутов и список ошибочных строк.

# Добавление маршрута

Производится выбор: добавить элемент в конец списка, в начало или по индексу. При вводе нового маршрута производится проверка корректности введённых данных. Если данные некорректны – пользователь возвращается в меню.

**void** addEnd(List \*\*startList, MARSH temporaryRoute);

При добавлении элемента в конец списка, элемент добавляется в конец списка, выводится сообщение об успешном добавлении, выводится список на консоль.

**void** addStart(List \*\*startList, MARSH temporaryRoute);

При добавлении элемента в начало списка, элемент добавляется в начало списка, выводится сообщение об успешном добавлении, выводится список на консоль.

**bool** addIndex(List \*\*startList, MARSH temporaryRoute, **int** index);

При добавлении элемента по индексу, пользователю предлагается ввести индекс, производится проверка на правильность введённого числа (число, состоящее из цифр от 0 до 9). Если индекс некорректен, пользователю предлагается ввести его заново.

Если вставить элемент по индексу не удаётся (количество элементов в списке меньше, чем индекс), выводится сообщение о том, что индекс некорректен. Индекс вводится до тех пор, пока не будет введён правильно.

Далее элемент добавляется на позицию (индекс – 1), считаем, что первый элемент в списке занимает позицию 0, далее выводится сообщение об успешном добавлении, выводится список на консоль.

# Удаление маршрута

**bool** deleteEl(List \*\*startList, MARSH temporaryRoute, **const** **int** status);

Список выводится на консоль.

Производится выбор: удалить все маршруты по названию начального пункта, или удалить все маршруты по названию конечного пункта, или удалить маршрут с конкретным номером. При вводе выбранного поля производится проверка корректности введённых данных. Если данные некорректны – пользователь повторяет ввод, пока данные не будут корректны.

Если данные корректно введены, производится поиск по выбранному полю и удаляются элементы, удовлетворяющие поиску (в случае с удалением по номеру маршрута удаляется единственный элемент). Выводится сообщение об успешном удалении, выводится список на консоль.

Если не было найдено элементов, удовлетворяющих условиям поиска, выводится сообщение об ошибке. Пользователь возвращается в меню.

# Вывод списка на экран

**void** print\_List(List \*stList);

Функция выводит на экран все элементы из списка, с разделителем между полями – тремя пробелами.

Пример выполнения:

ТРЦ Фестиваль Поселок Крим 120

Улица Лесная Улица Горького 44

Кинотеатр Мир Северная 40

# Поиск

**bool** findEl(List \*stList, List \*\*startFindList, string firstOrEnd);

Пользователь вводит строку символов, по которой будет осуществляться поиск. Производится проверка на корректность ввода, аналогичная для начального/конечного пункта маршрута. Если данные введены некорректно, пользователю предлагается ввести их еще раз.

Производится поиск, в результате которого все маршруты, начальный или конечный пункт которых содержит в себе данную строку, добавляются в односвязный список найденных маршрутов.

Список найденных маршрутов выводится на консоль.

Если элементов, удовлетворяющих условиям поиска, в списке нет, то выводится сообщение об отсутствии подходящего элемента в списке. Если список для поиска был пуст, выводится сообщение об его пустоте.

# Сортировка

**void** sortList(List \*\*startList, **int** status);

Список до сортировки выводится на консоль.

Выполняется сортировка по начальному пункту маршрута (по алфавиту; пробел «больше» чем дефис).

Если список пуст, или в нём 1 элемент, выводится сообщение, что список негоден для сортировки. Пользователь возвращается в меню.

После выполнения сортировки, выводится сообщение об успешном выполнении, список выводится на экран. Пользователь возвращается в меню.

**Редактирование**

**void** redactEl(List \*stList, string routeNumb);

Пользователю предлагается ввести номер маршрута, который он хочет отредактировать. Производится проверка на правильность ввода номера маршрута. Если номер маршрута введён некорректно, пользователь вводит его, пока не введёт корректный номер. Производится проверка на наличие маршрута с данным номером в списке. Если маршрут не найден – пользователь возвращается в меню.

Производится выбор: редактировать начальный пункт маршрута, редактировать конечный пункт маршрута, редактировать номер маршрута или закончить редактирование.

При выборе «закончить редактирование», пользователь направляется в меню.

При вводе данных для выбранного поля для редактирования, производится проверка корректности введённых данных. Ввод повторяется, пока не будут введены корректные данные.

Производятся: проверка на отличие введённых данных от тех, что были, проверка на совпадение начального и конечного пунктов (при редактировании начального/конечного пункта маршрута), проверка на уникальность будущего маршрута. Если какая-то проверка не пройдена, пользователь получает соответствующее сообщение о провале проверки. Пользователь вводит данные, пока они не пройдут все проверки.

Далее маршрут редактируется, пользователь получает сообщение о том, что редактирование произведено, обновлённый список маршрутов выводится в консоль, и пользователь возвращается в меню для редактирования. Редактирование продолжается, пока пользователь не выберет «Закончить редактирование».

**Удаление списка**

**void** deleteList(List \*\*startList);

Задаётся вопрос для потверждения удаления всего списка. Если пользователь подтверждает, выводится сообщение об успешном удалении списка, список удаляется.

Если пользователь не подтверждает, выводится сообщение о том, что список не удалён, пользователь возвращается в меню, список остается таким, каким был.

**Сохранение данных в файл**

**int** writingFile(List \*stList);

Данные из линейного списка сохраняются в файл формата .txt, выводится сообщение об успехе на консоль. Если список пуст, выводится соответствующее сообщение, файл становится пустым.

# Выход из программы

Пользователь выходит из программы. Список удаляется.

1. Описание пользовательского интерфейса

При запуске программы на экран выводится консольное приложение с меню пользователя. При нажатии на клавиатуре на определенную цифру выполняется соответствующая функция.

Список команд:

1. Выход
2. Загрузить данные из файла
3. Добавить маршрут
4. Удалить маршрут
5. Показать список
6. Найти маршрут
7. Отсортировать список
8. Редактировать маршрут
9. Удалить весь список
10. Выгрузить данные в файл

Пользователю дается возможность выполнения задач в любом порядке.

1. Результаты тестирования программы
   1. Словесное описание

|  |  |  |  |
| --- | --- | --- | --- |
| № | Действие | Входные данные | Результат |
| 1 | Добавление записи (цифра 2) | Вводятся значения  всех полей, информация о том, куда добавить – в начало, в конец, по индексу | Происходит добавление.  Вывод сообщения о добавлении.  Вывод списка. |
| 2 | Редактирование  (цифра 7) | Вводится значение поля номера маршрута. | Изменяются значения всех полей записи.  Вывод сообщения о редактировании.  Вывод списка на экран. |
| 3 | Удаление записи  (цифра 3) | Вводится значение поля начального пункта, или конечного пункта, или номера маршрута | Происходит удаление. Вывод списка на экран. |

|  |  |  |  |
| --- | --- | --- | --- |
| № | Действие | Входные данные | Результат |
| 4 | Сортировка (цифра 6) | — | Происходит сортировка.  Вывод отсортированного списка на экран. |
| 5 | Поиск маршрутов по названию начального/конечного пункта (цифра 5) | Вводится строка, по которой осуществляется поиск | Происходит поиск элемента(ов).  Найденные элементы выводятся на экран. |
| 6 | Загрузка данных из файла (цифра 1) | Выбор: перезаписать список, или добавить элементы в существующий | Происходит загрузка данных.  Вывод сообщения о успешной загрузке. |
| 7 | Сохранение данных в файл (цифра 9) | — | Происходит сохранение данных. Вывод сообщения о успешном сохранении. |
| 8 | Вывод списка  (цифра 4) | — | Весь список выводится  на экран. |
| 9 | Удаление списка (цифра 8) | Выбор: продолжить удаление или остановить | Список удаляется. Вывод сообщения об успешном удалении. |
| 10 | Завершение программы (цифра 0) | — | Удаление списка. Выход из программы. |

* 1. Тестовые данные

|  |  |  |  |
| --- | --- | --- | --- |
| № | Действие | Входные данные | Результат |
| 1 | Добавление записи (цифра 2) | 2  1  Кинотеатр Мир Северная  40 | Элемент добавлен.  Фестиваль Поселок 120  Улица Лесная Горький 44  Улица Лесная Сладкий 35  Кинотеатр Мир Северная 40 |
| 2 | Редактирование  (цифра 7) | 7  44  1  Кировский | Редактирование произведено.  Фестиваль Поселок 120  Кировский Горький 44  Улица Лесная Сладкий 35  Кинотеатр Мир Северная 40 |
| 3 | Удаление записи  (цифра 3) | 3  2  Сладкий | Список до удаления:  Улица Лесная Сладкий 35  Список после удаления:  Список пуст! |
| 4 | Сортировка  (цифра 6) | 6 | Кировский Горький 44  Улица Лесная Сладкий 35  Кинотеатр Мир Северная 40 Список отсортирован!  Кинотеатр Мир Северная 40 Кировский Горький 44  Улица Лесная Сладкий 35 |
| 5 | Поиск маршрутов по названию начального/конечного пункта (цифра 5) | 5  кий | Кировский Горький 44  Улица Лесная Сладкий 35 |

|  |  |  |  |
| --- | --- | --- | --- |
| № | Действие | Входные данные | Результат |
| 6 | Загрузка данных из файла (цифра 1) | 1 | Данные успешно выгружены в список, который выглядит следующим образом:  Фестиваль Поселок 120  Улица Лесная Горький 44  Улица Лесная Сладкий 35 |
| 7 | Сохранение данных в файл (цифра 9) | 9 | Данные успешно записаны в файл |
| 8 | Вывод списка  (цифра 4) | 4 | Кинотеатр Мир Северная 40 Кировский Горький 44  Улица Лесная Сладкий 35 |
| 9 | Удаление списка (цифра 8) | 8 | Список удалён! |
| 10 | Завершение программы (цифра 0) | 0 | — |

* 1. Скриншоты выполнения тестовых данных

****

Рисунок 1 – Главное меню

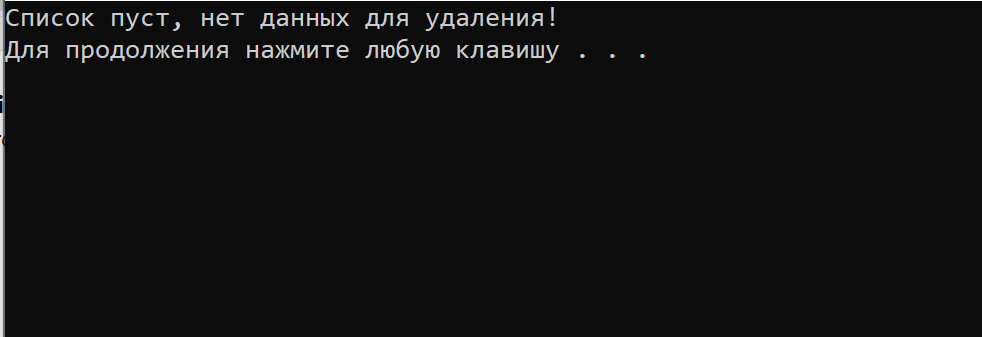


Рисунок 2 – Проверка удаления пустого списка

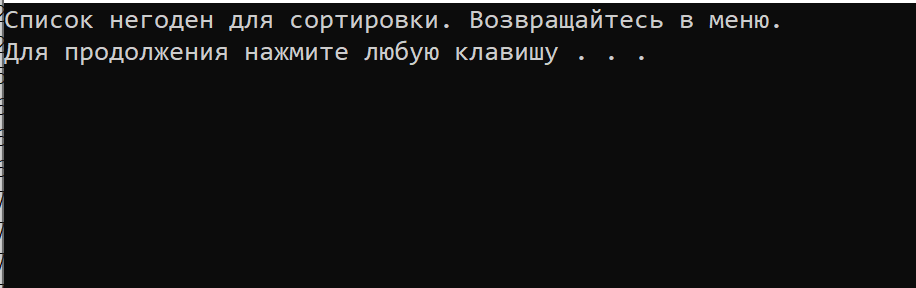


Рисунок 3 – Проверка сортировки пустого списка

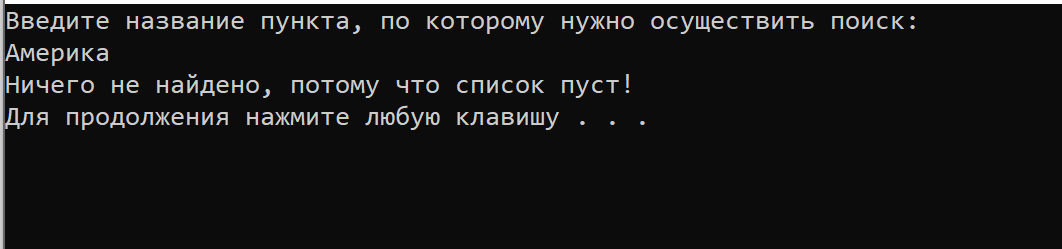


Рисунок 5 – Проверка поиска по пустому списку

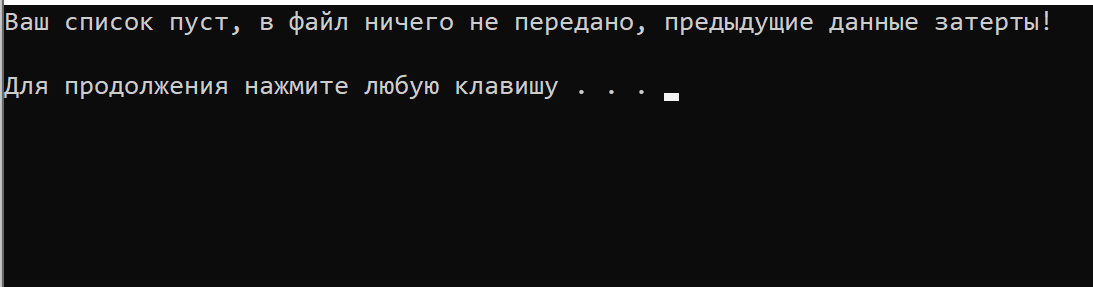


Рисунок 6 – Проверка сохранения пустого списка

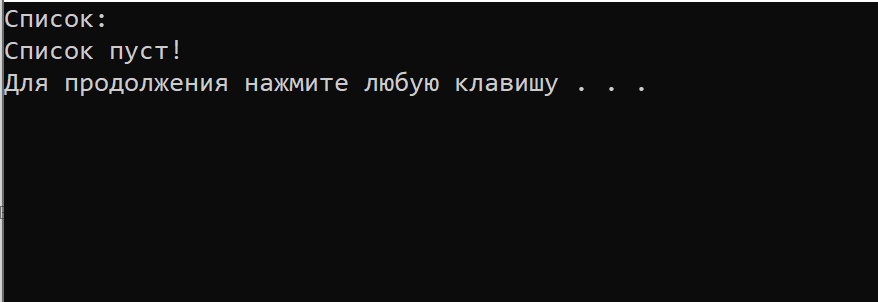


Рисунок 7 – Проверка вывода пустого списка

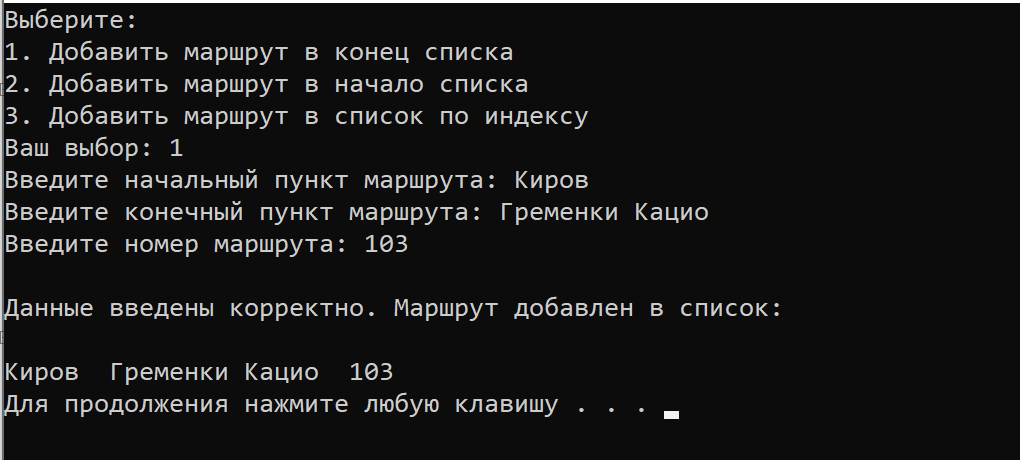


Рисунок 8 – Добавление записи в конец списка

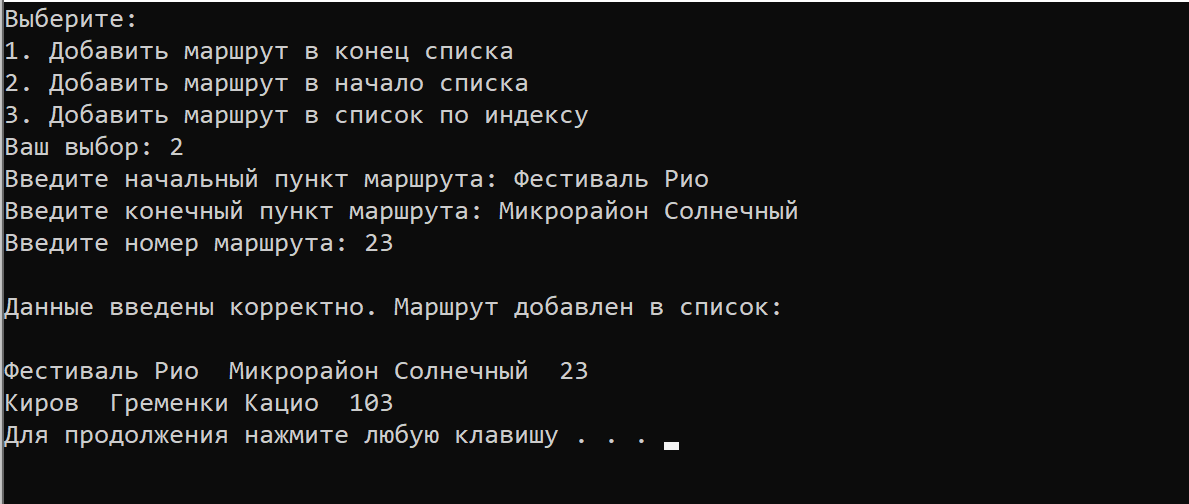


Рисунок 9 – Добавление записи в начало списка

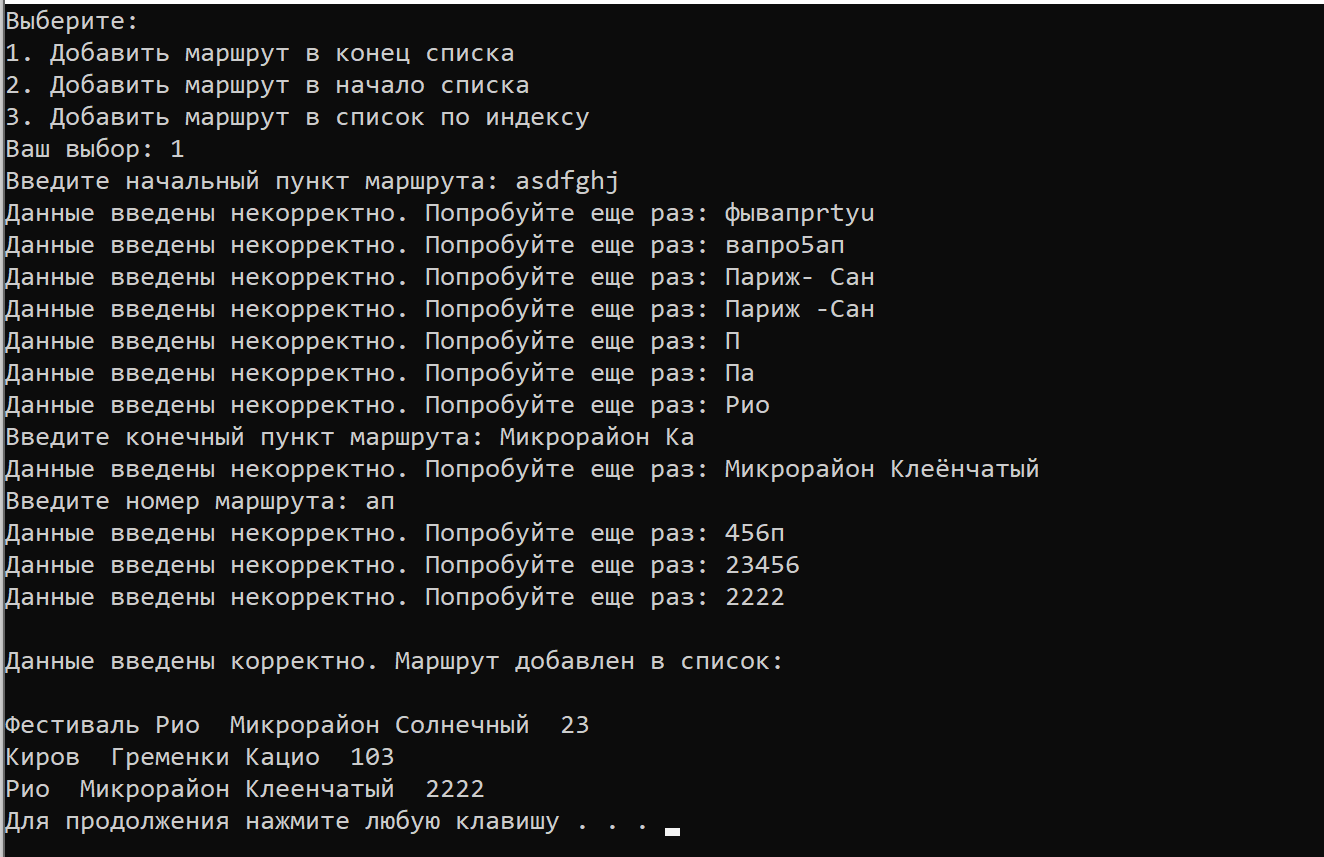


Рисунок 10 – Добавление записи с проверкой корректности ввода

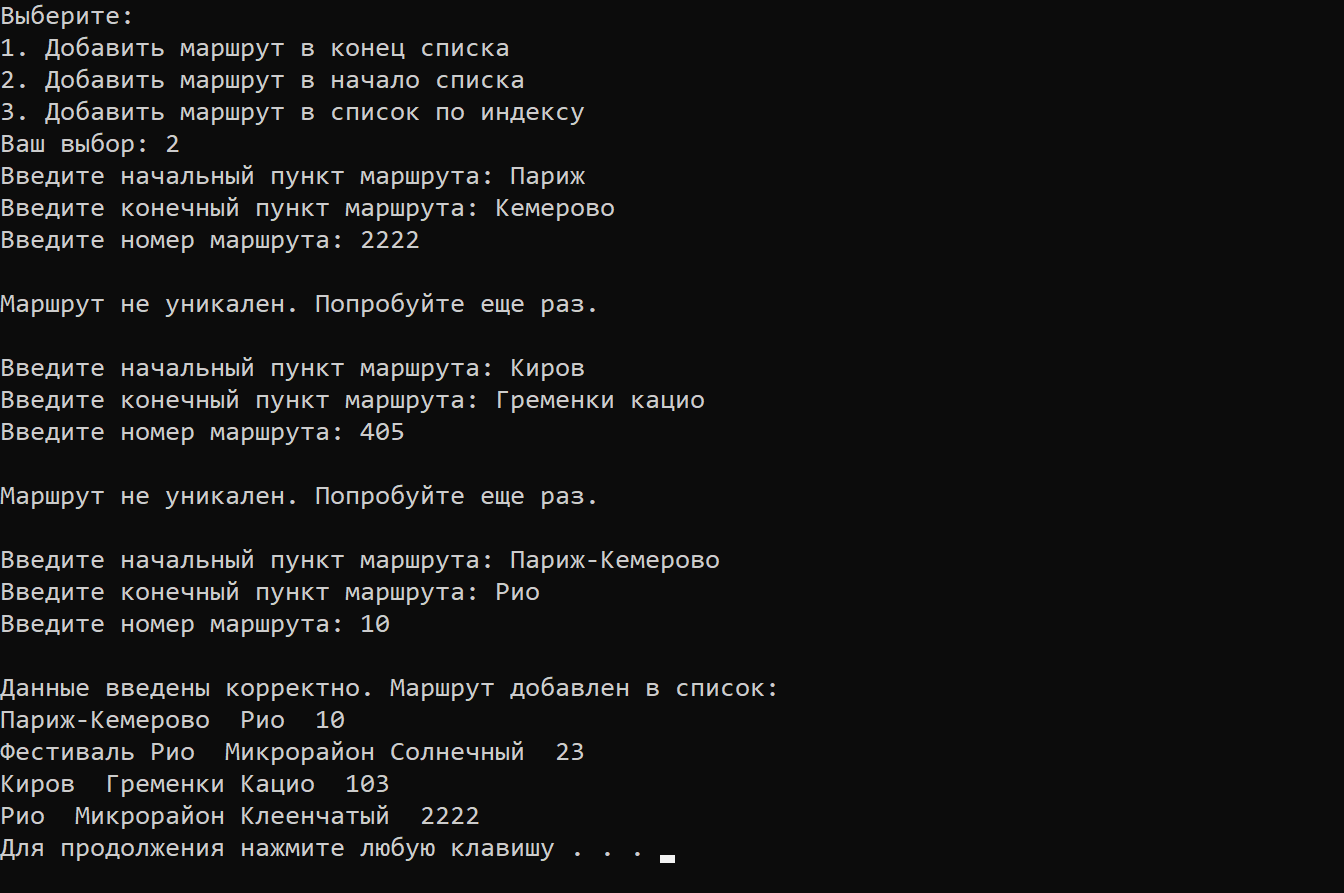


Рисунок 11 – Добавление записи с проверкой на уникальность

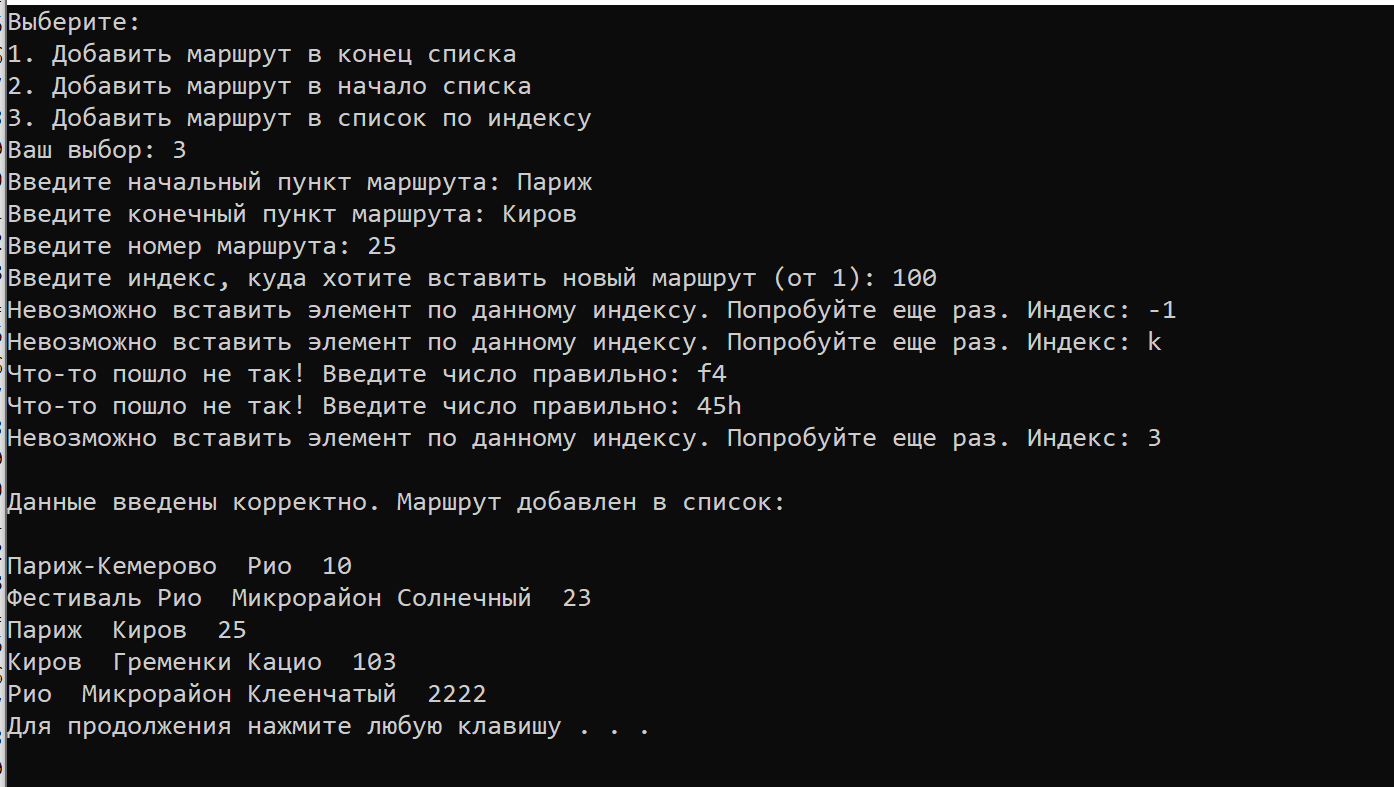


Рисунок 12 – Добавление записи по индексу в середину

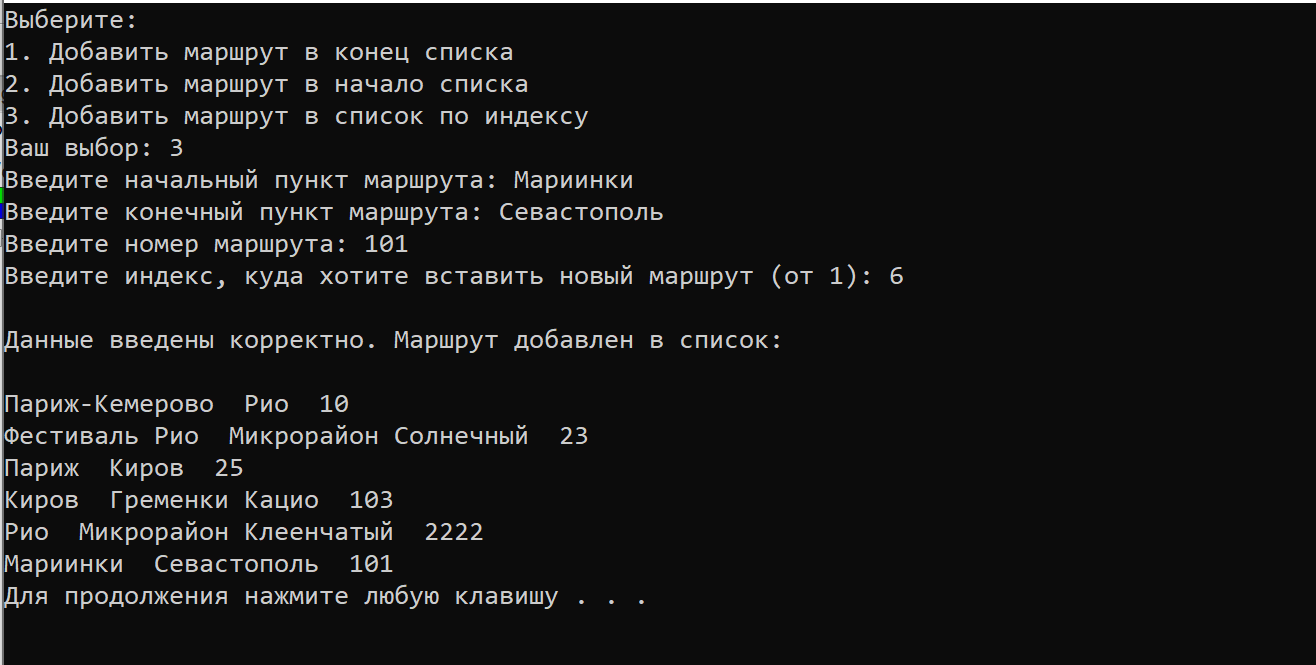


Рисунок 13 – Добавление записи по индексу в конец

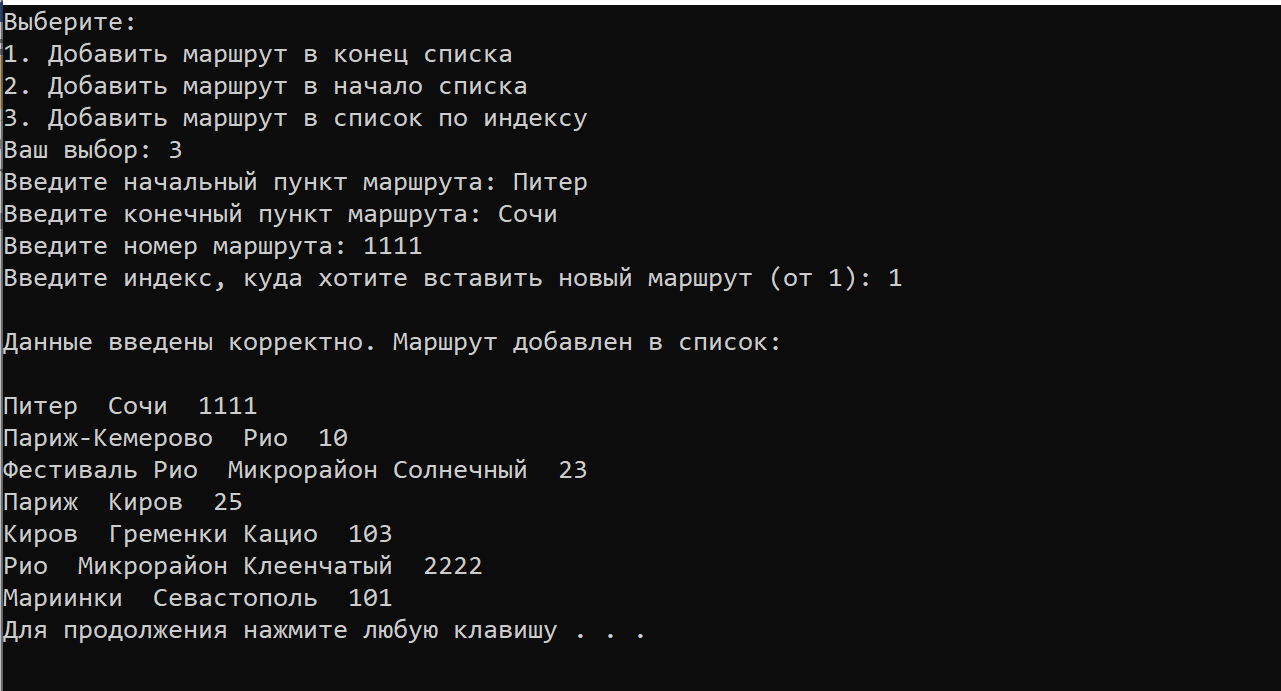


Рисунок 14 – Добавление записи по индексу в начало

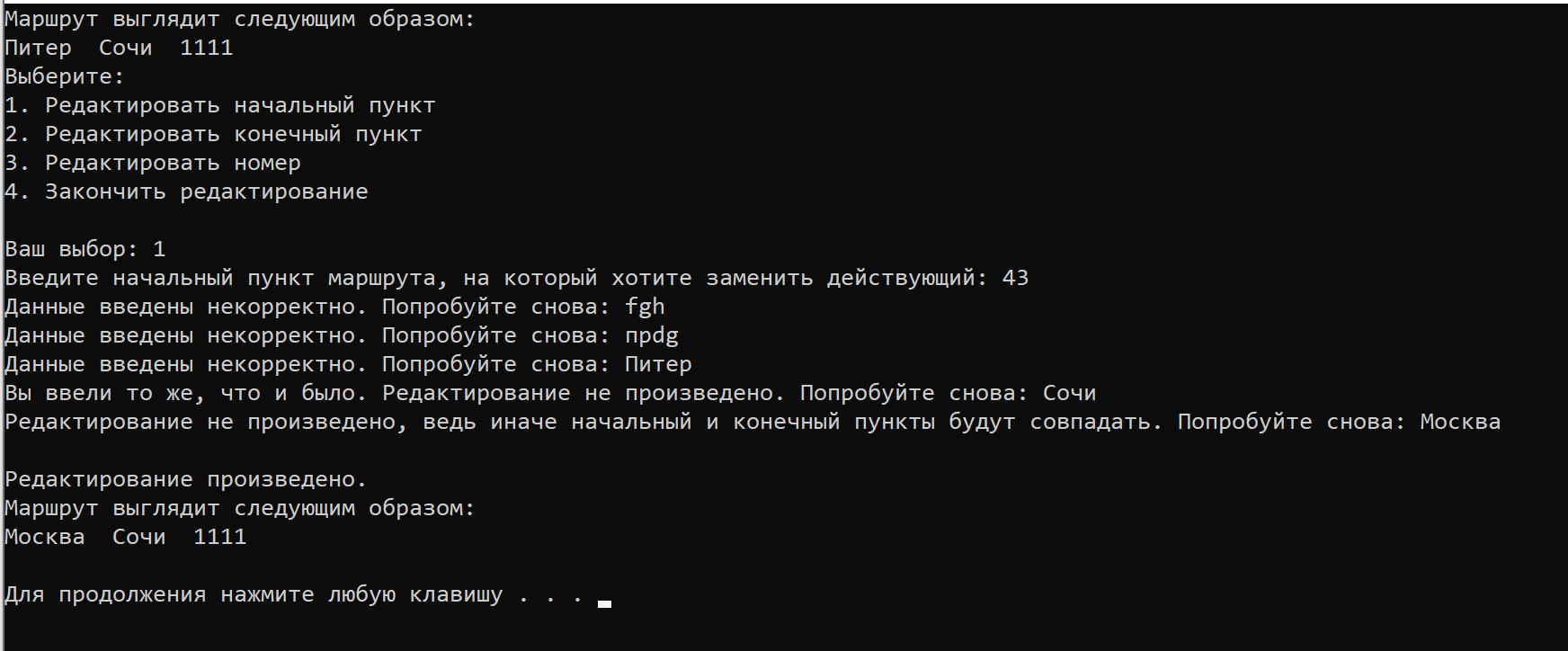


Рисунок 15 – Редактирование начального пункта с проверкой ввода

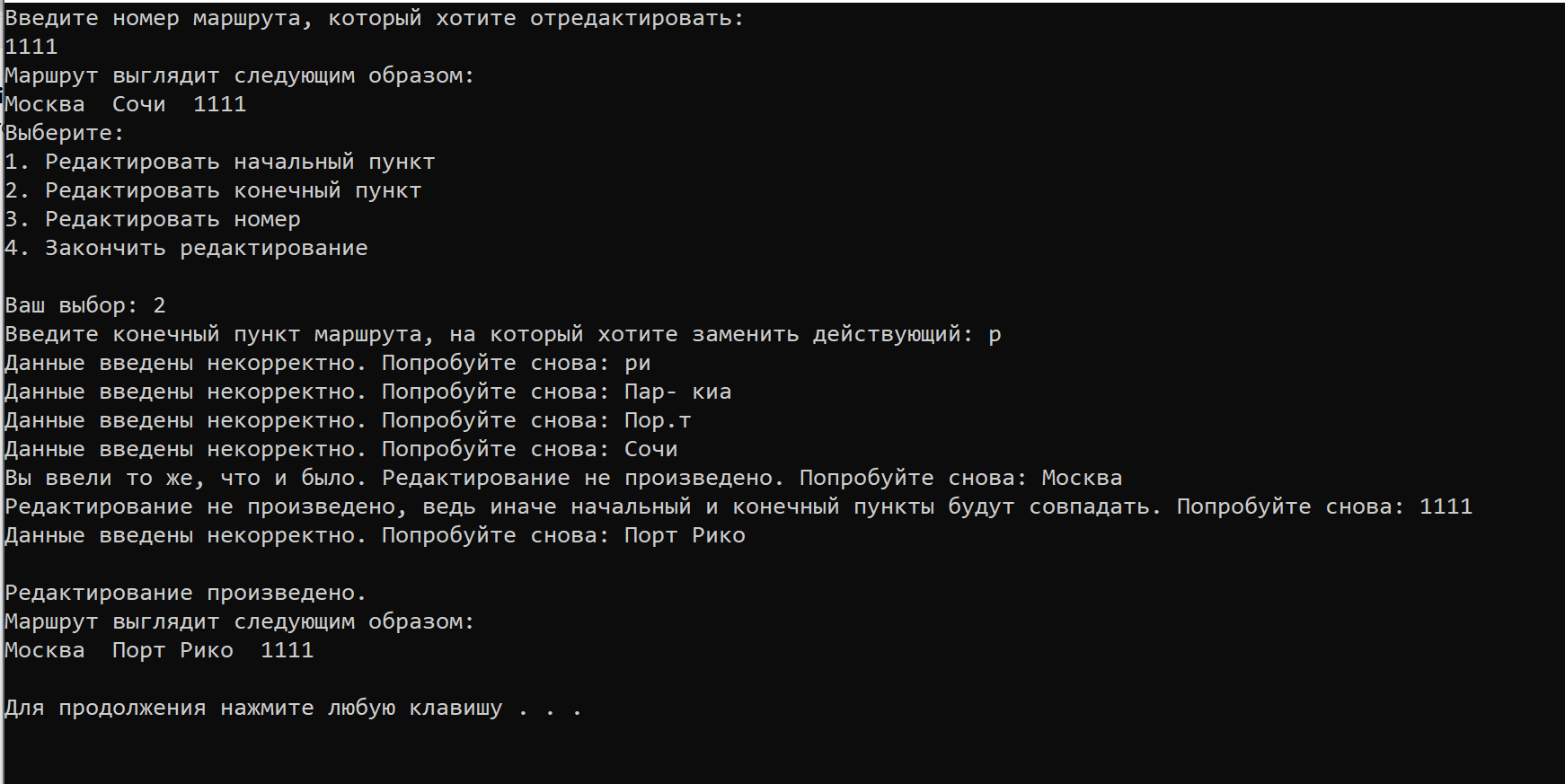


Рисунок 16 – Редактирование конечного пункта с проверкой ввода

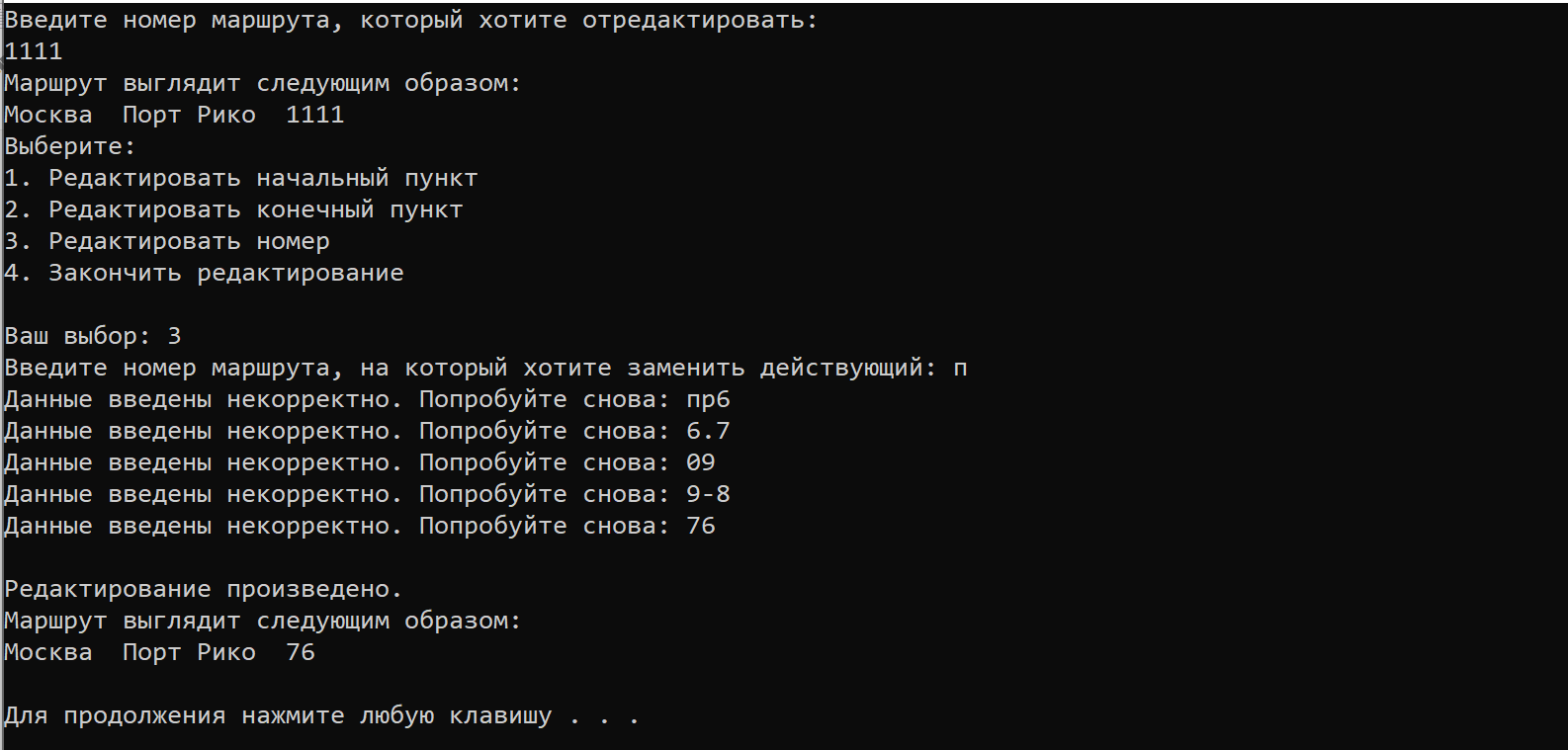


Рисунок 17 – Редактирование номера маршрута с проверкой корректности ввода

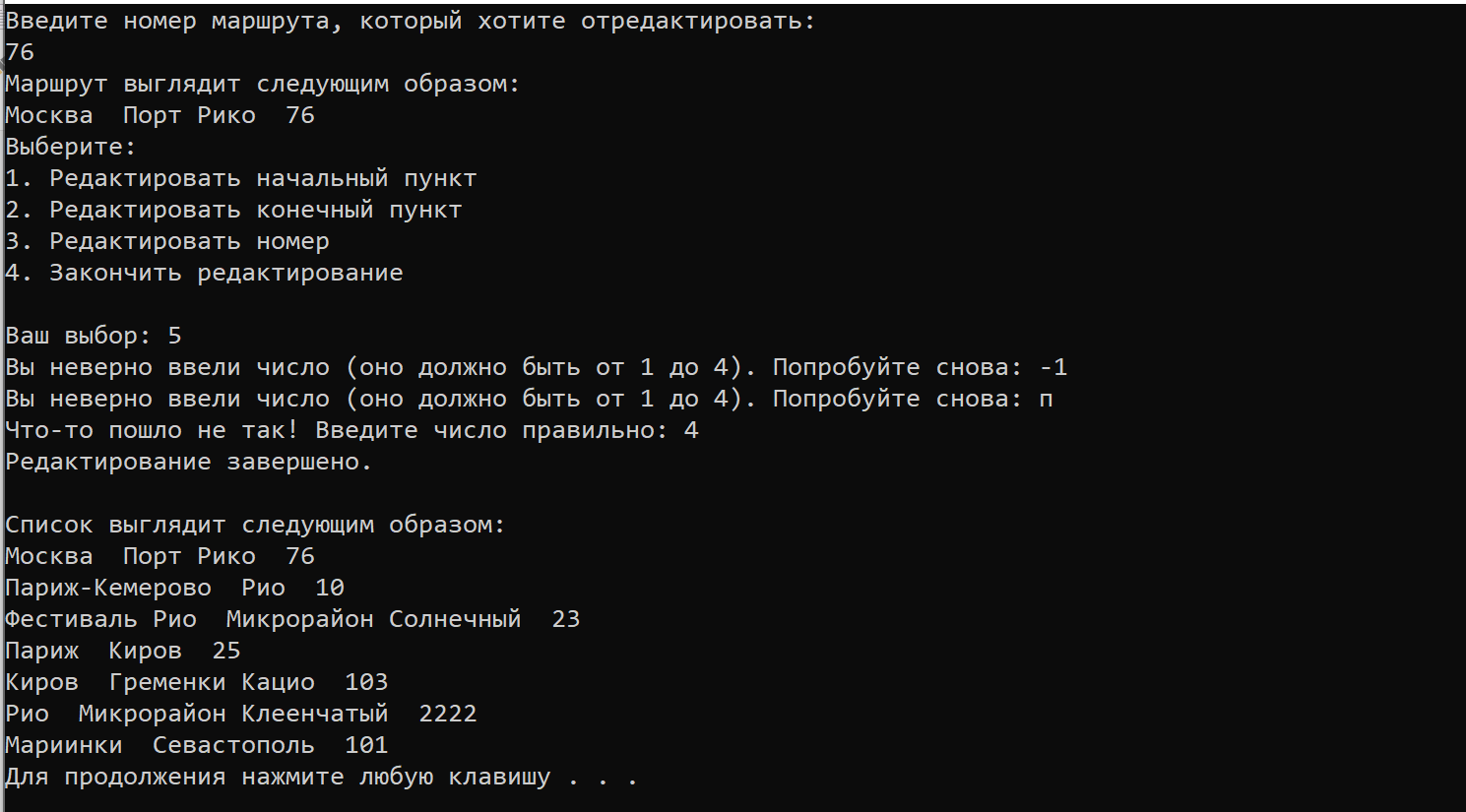


Рисунок 18 – Завершение редактирования с проверкой правильности выбора действия для редактирования

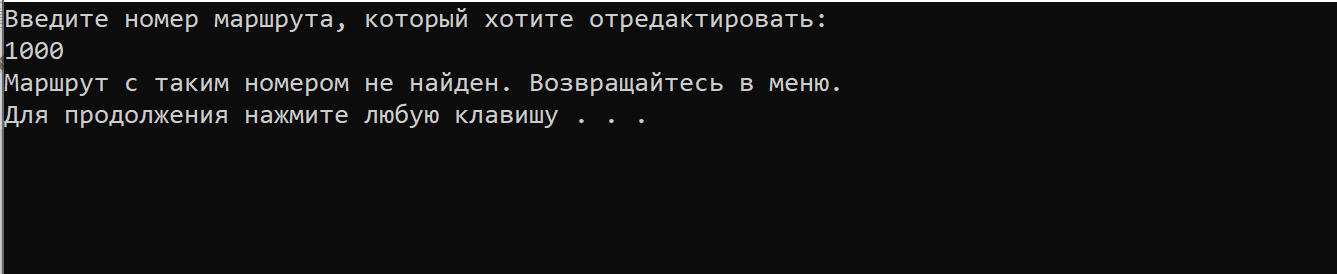


Рисунок 19 – Редактирование маршрута с несуществующим номером

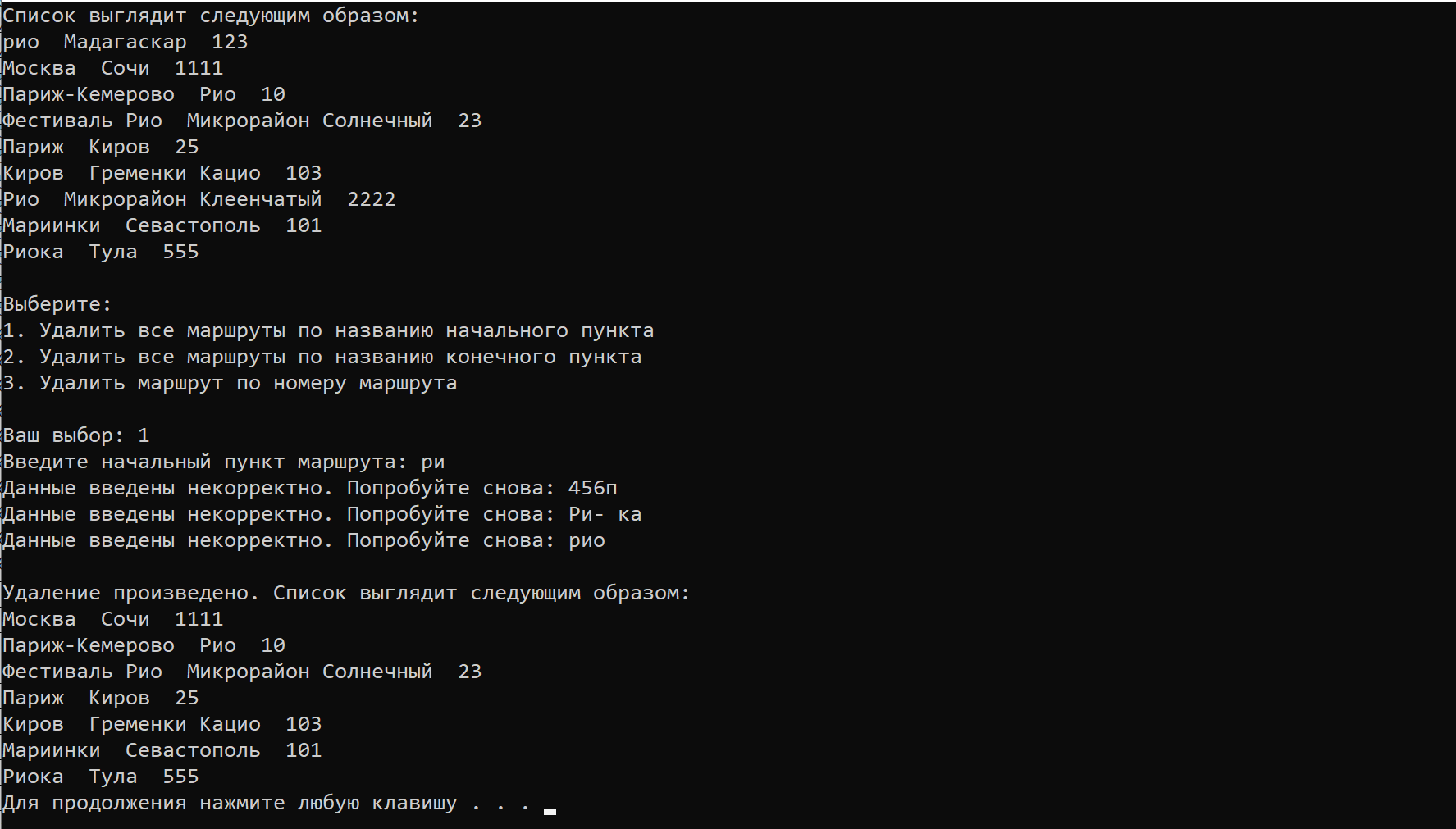


Рисунок 20 – Удаление записи по названию начального пункта с проверкой правильности ввода

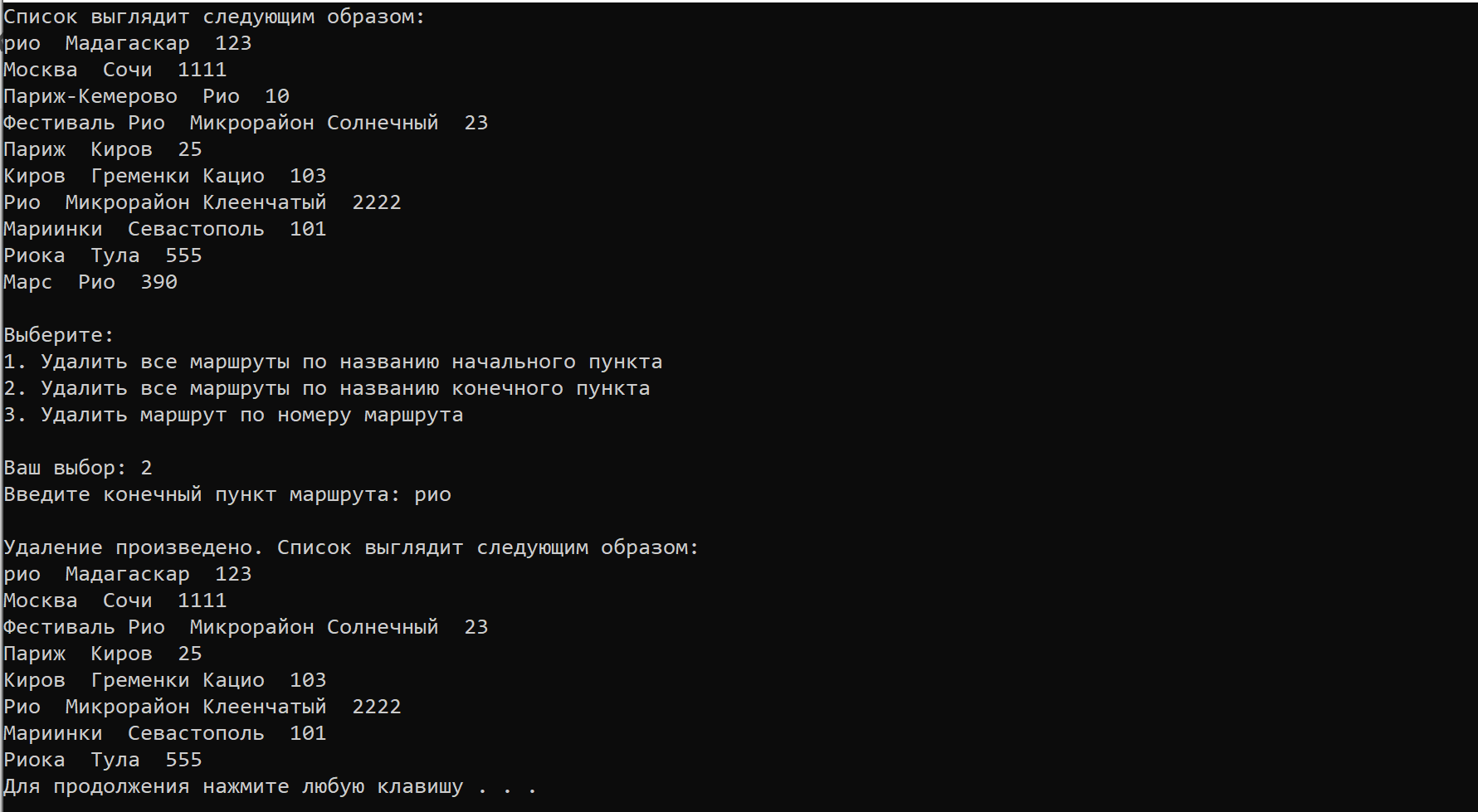


Рисунок 21 – Удаление записи по названию конечного пункта

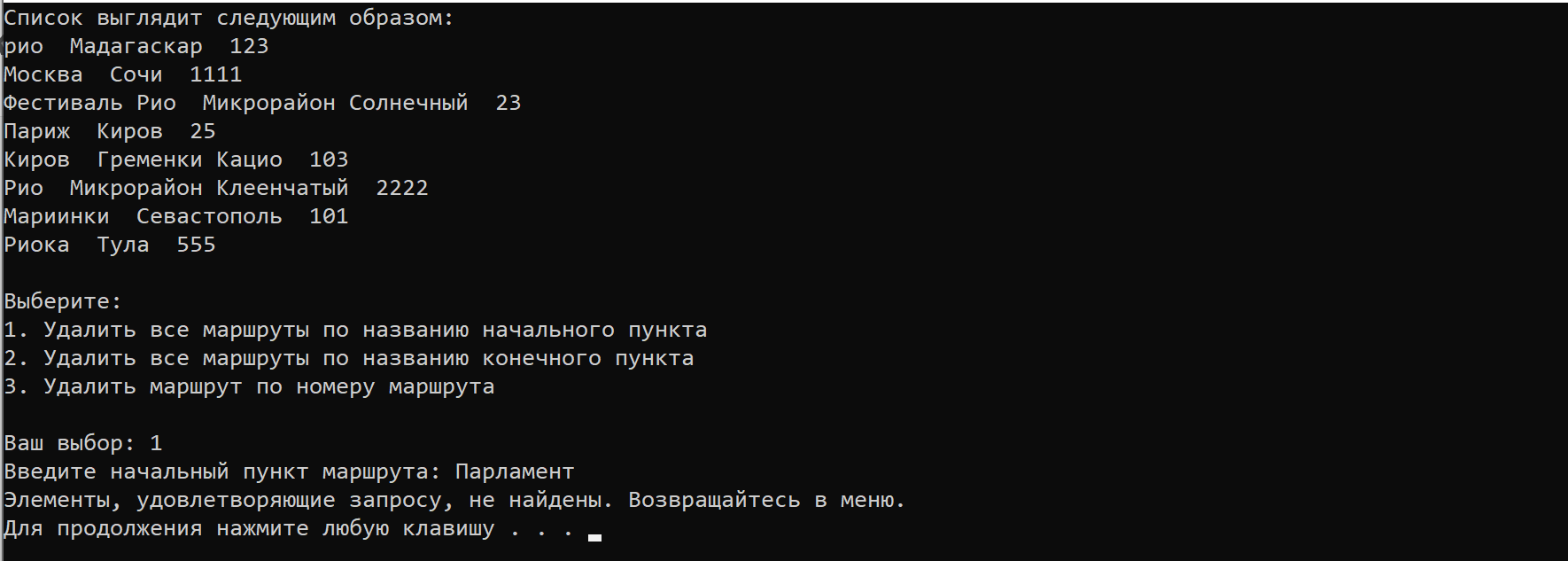


Рисунок 22 – Удаление несуществующей записи по названию начального пункта

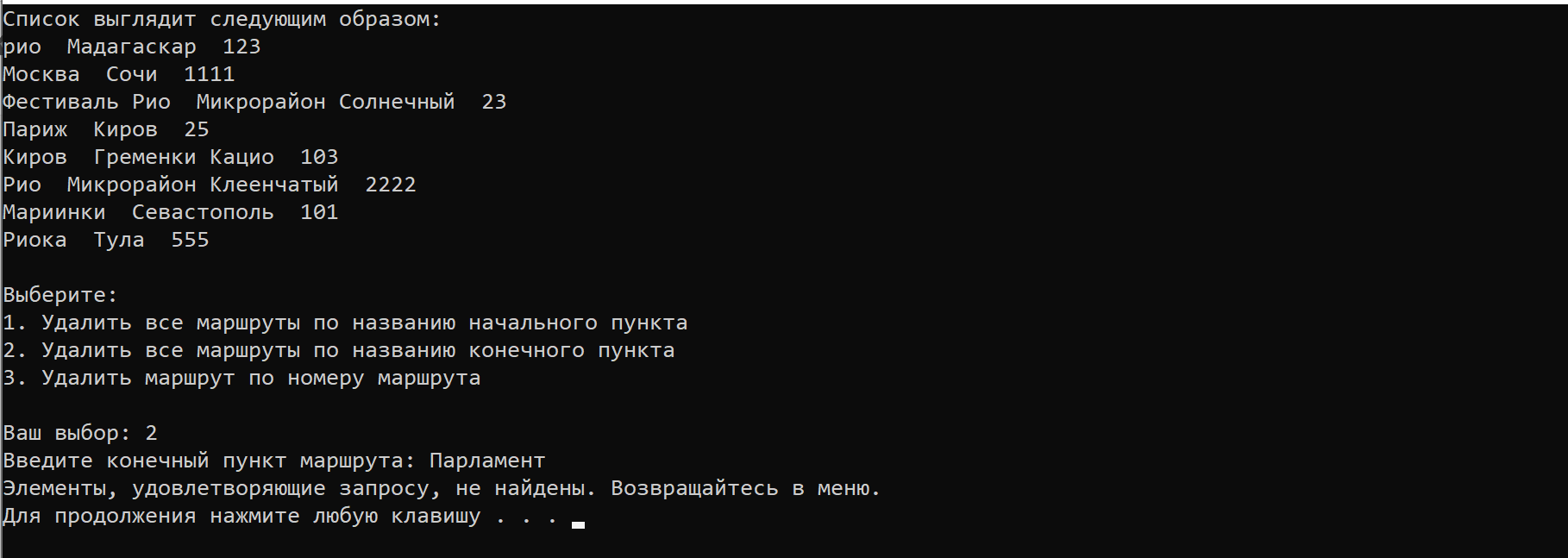


Рисунок 23 – Удаление несуществующей записи по названию конечного пункта

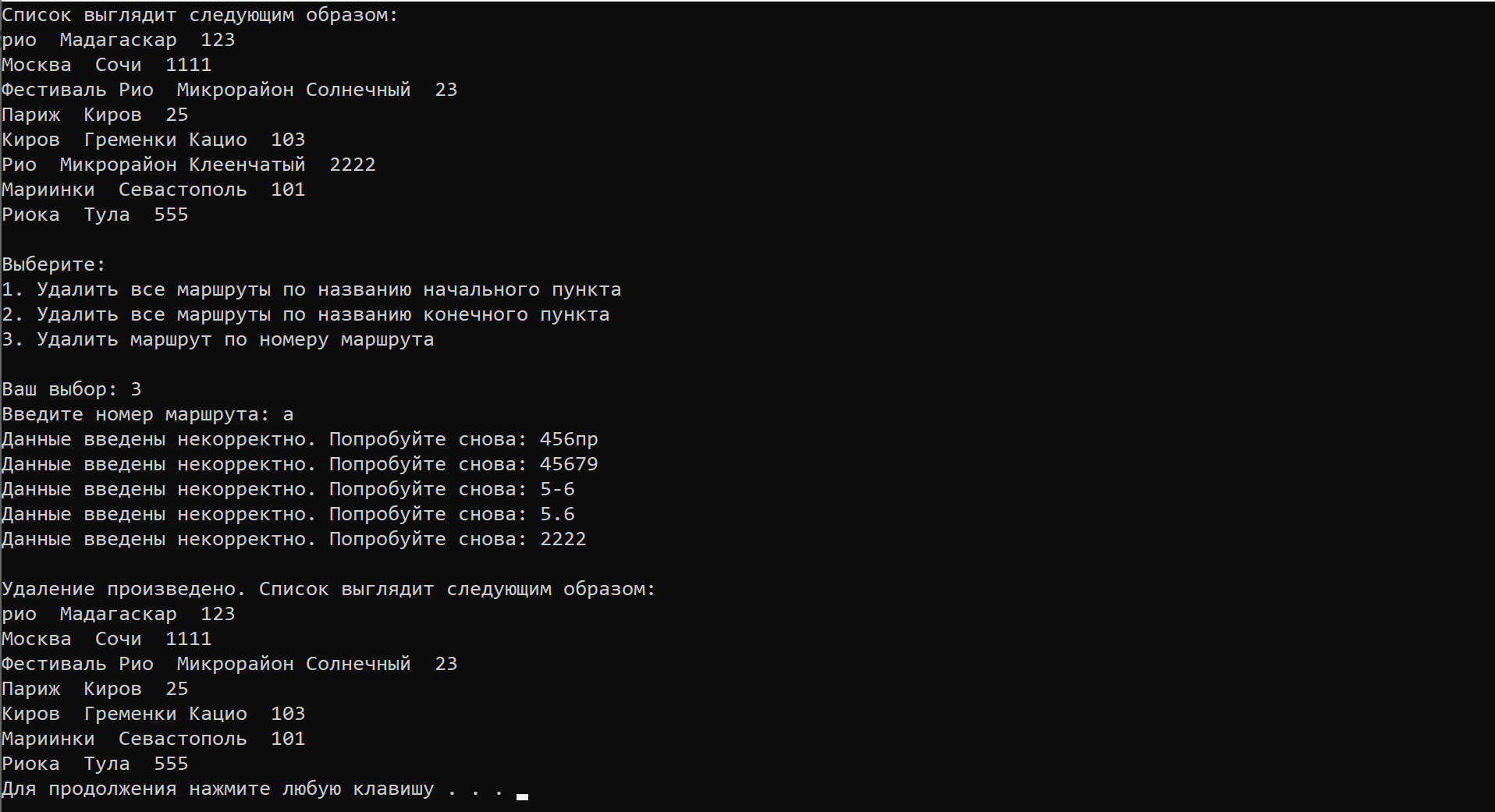


Рисунок 24 – Удаление записи по номеру маршрута с проверкой ввода

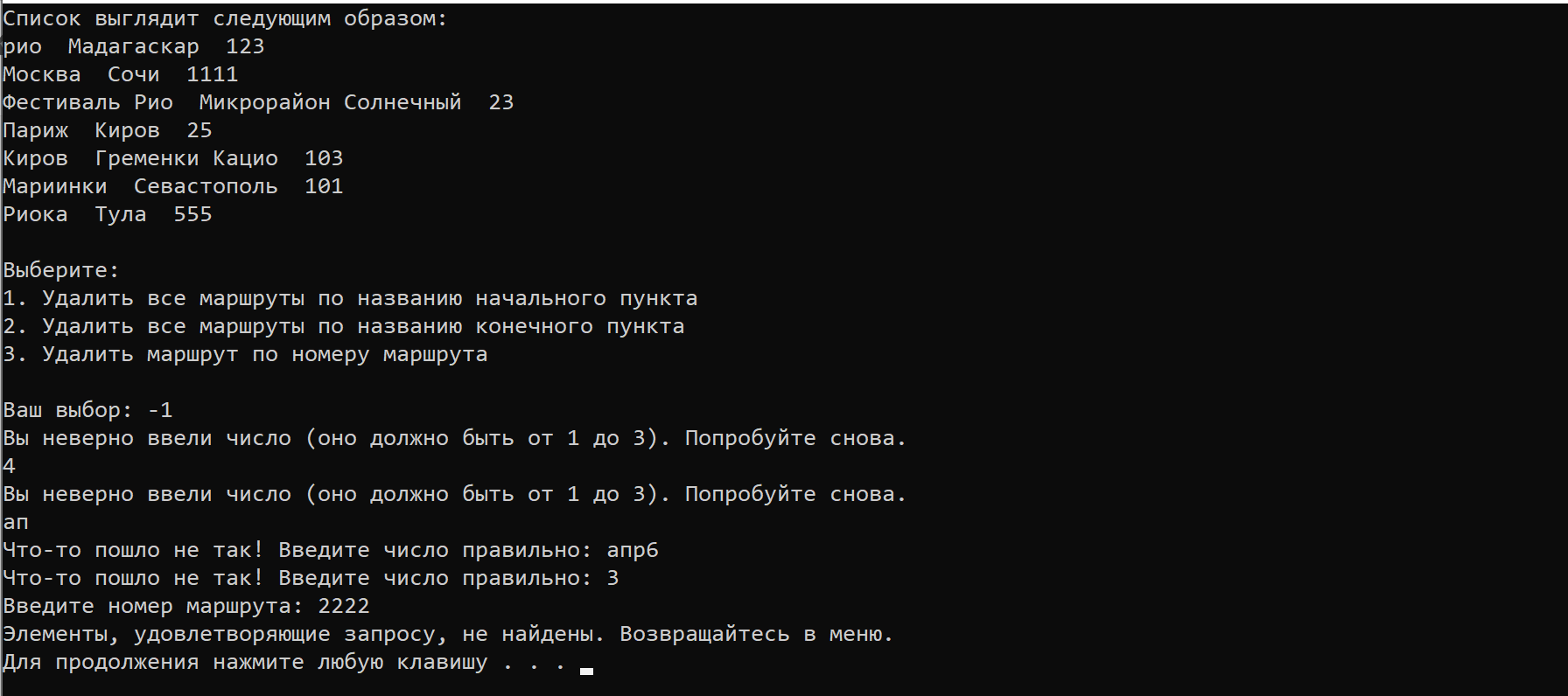


Рисунок 25 – Удаление несуществующей записи по номеру маршрута с проверкой на правильность введённого выбора

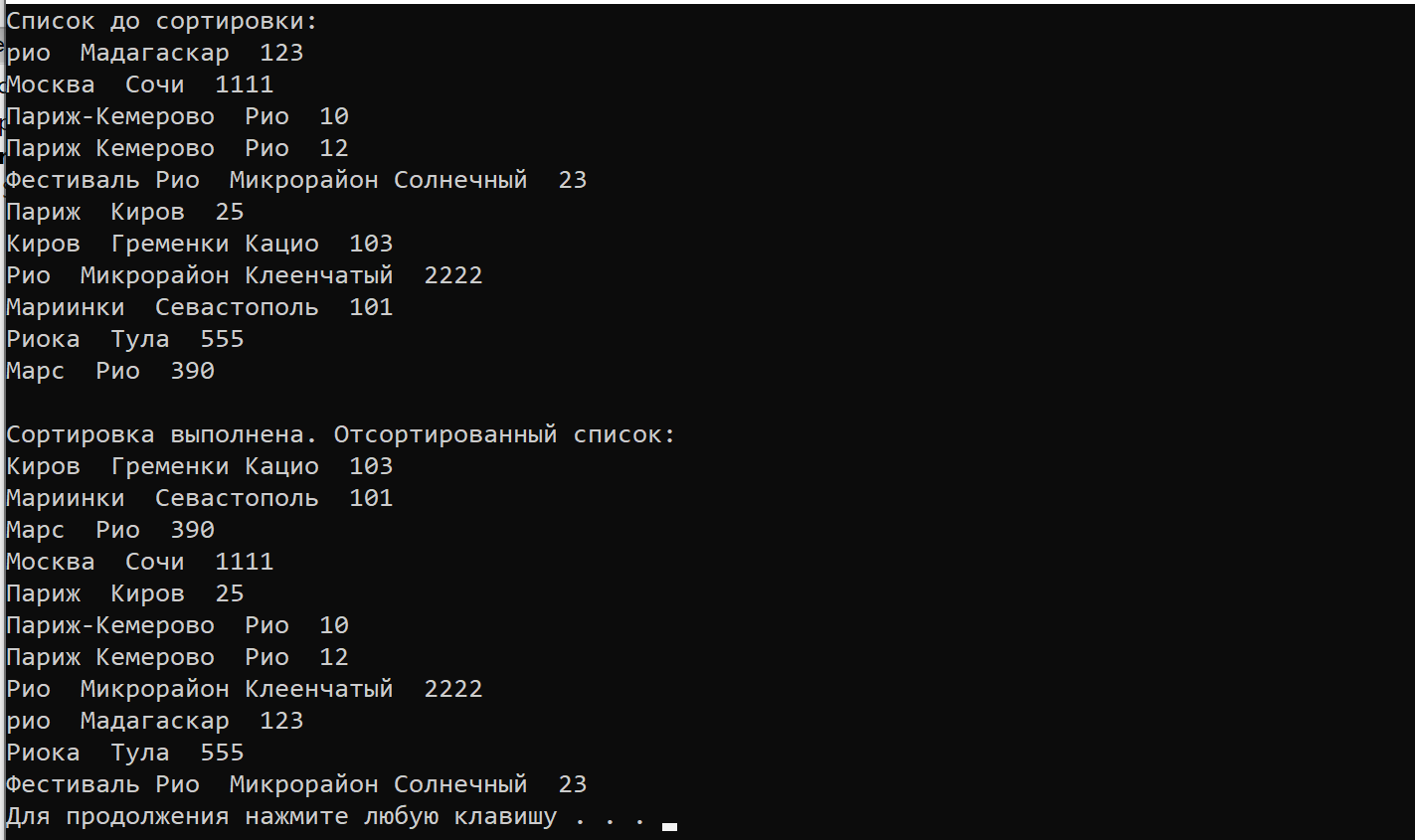


Рисунок 26 – Сортировка

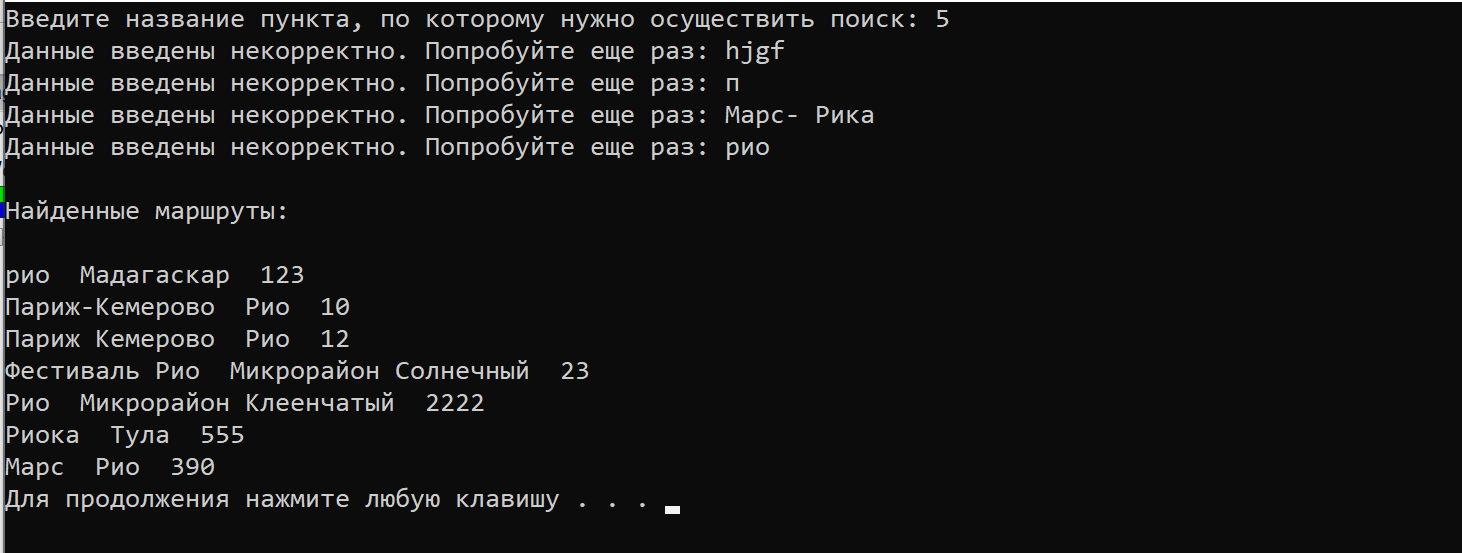


Рисунок 27 – Поиск маршрутов с проверкой на корректность данных

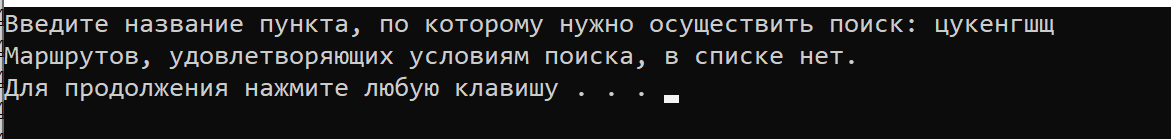


Рисунок 28 – Поиск несуществующего маршрута

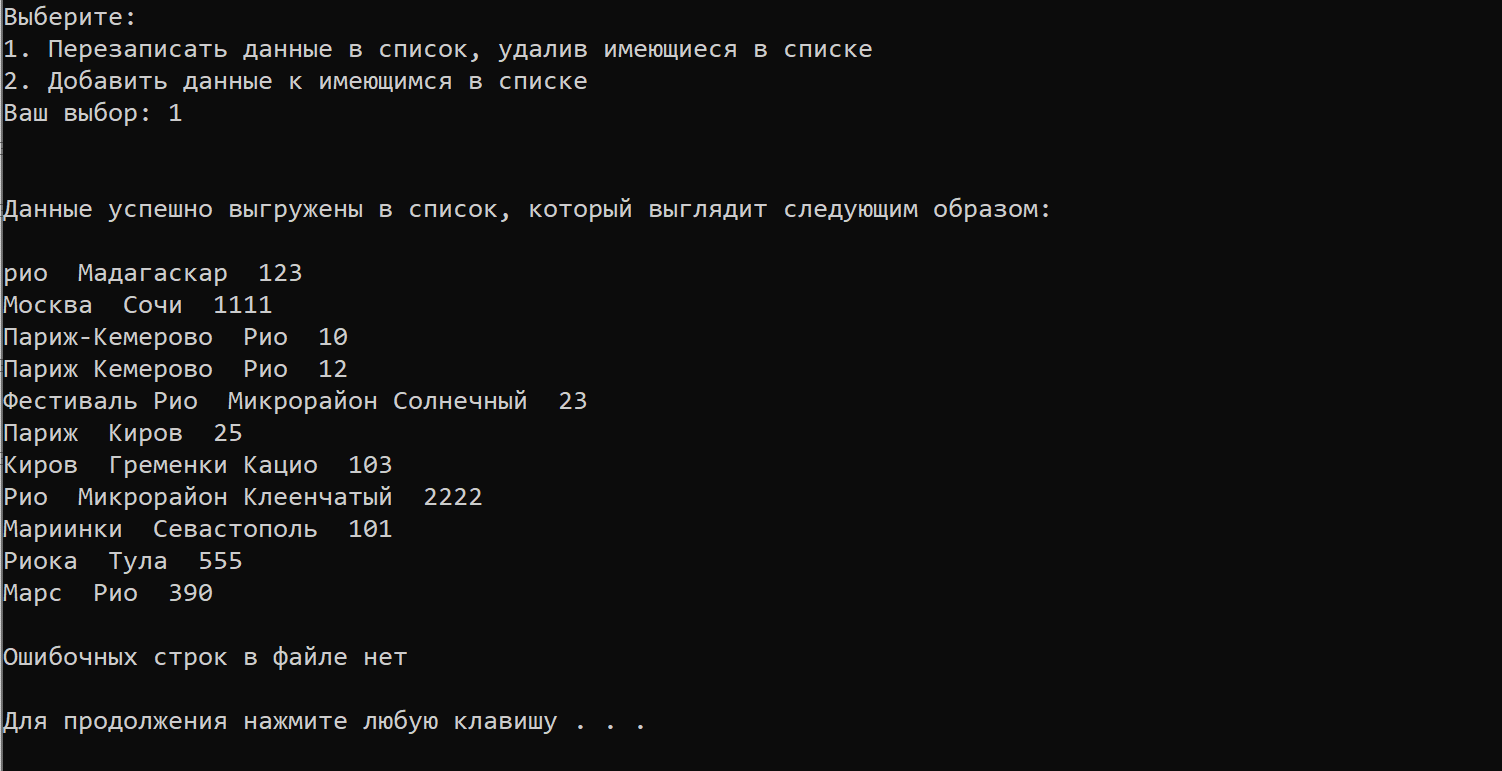


Рисунок 29 – Загрузка данных из файла, в котором нет ошибочных строк, при обновлении существующего списка

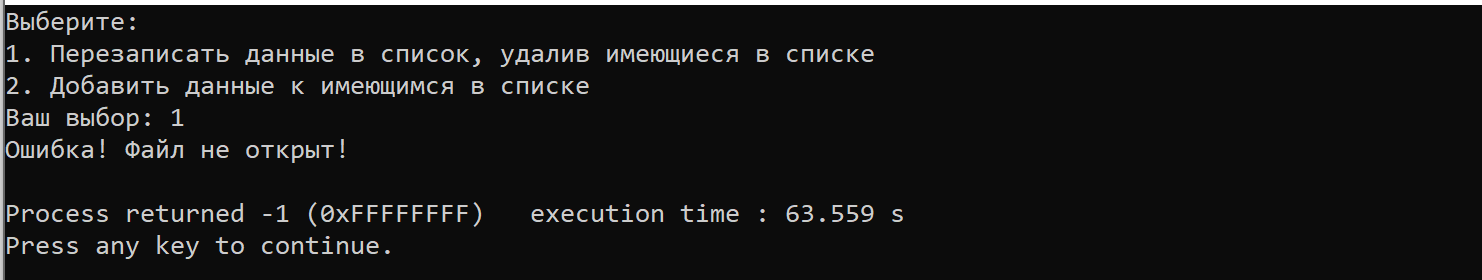


Рисунок 30 – Ошибка загрузки данных из файла при обновлении существующего списка

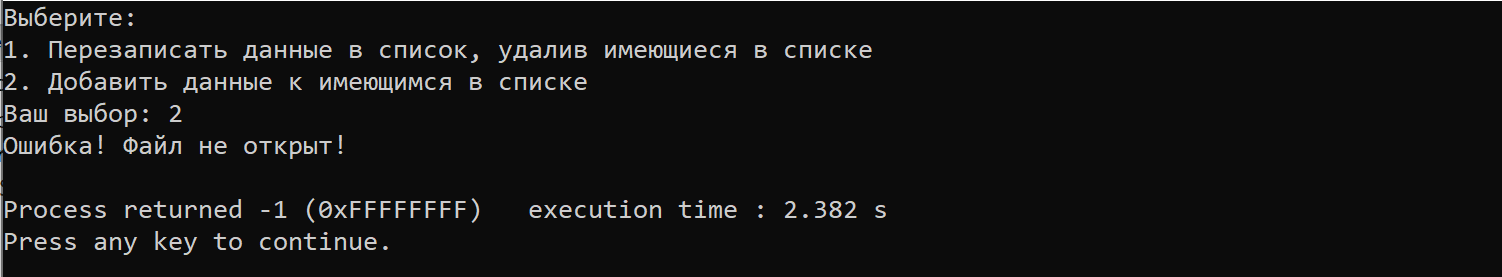


Рисунок 31 – Ошибка загрузки данных из файла при добавлении данных к имеющимся в списке

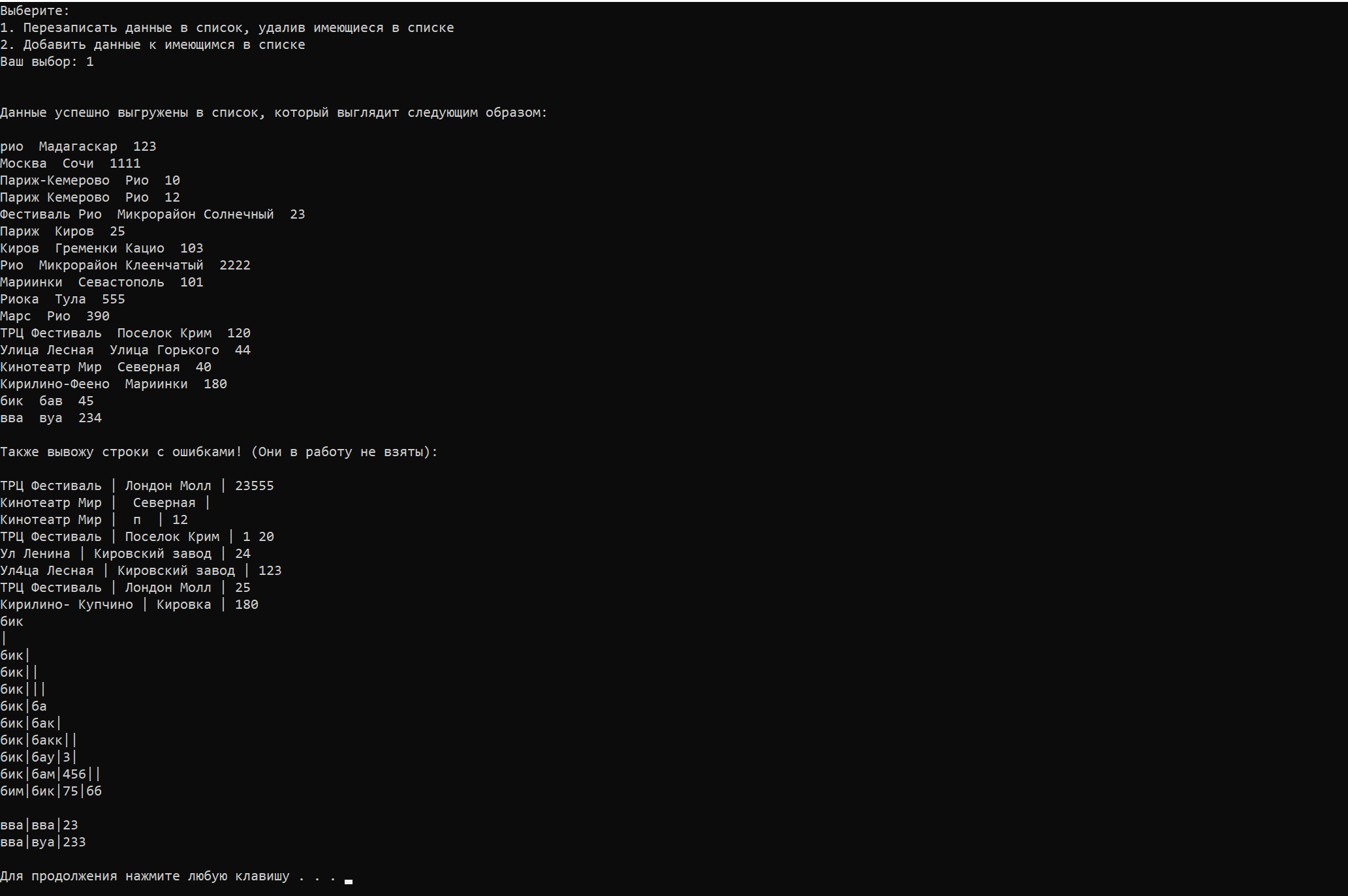


Рисунок 32 – Загрузка данных из файла, в котором есть ошибочные строки

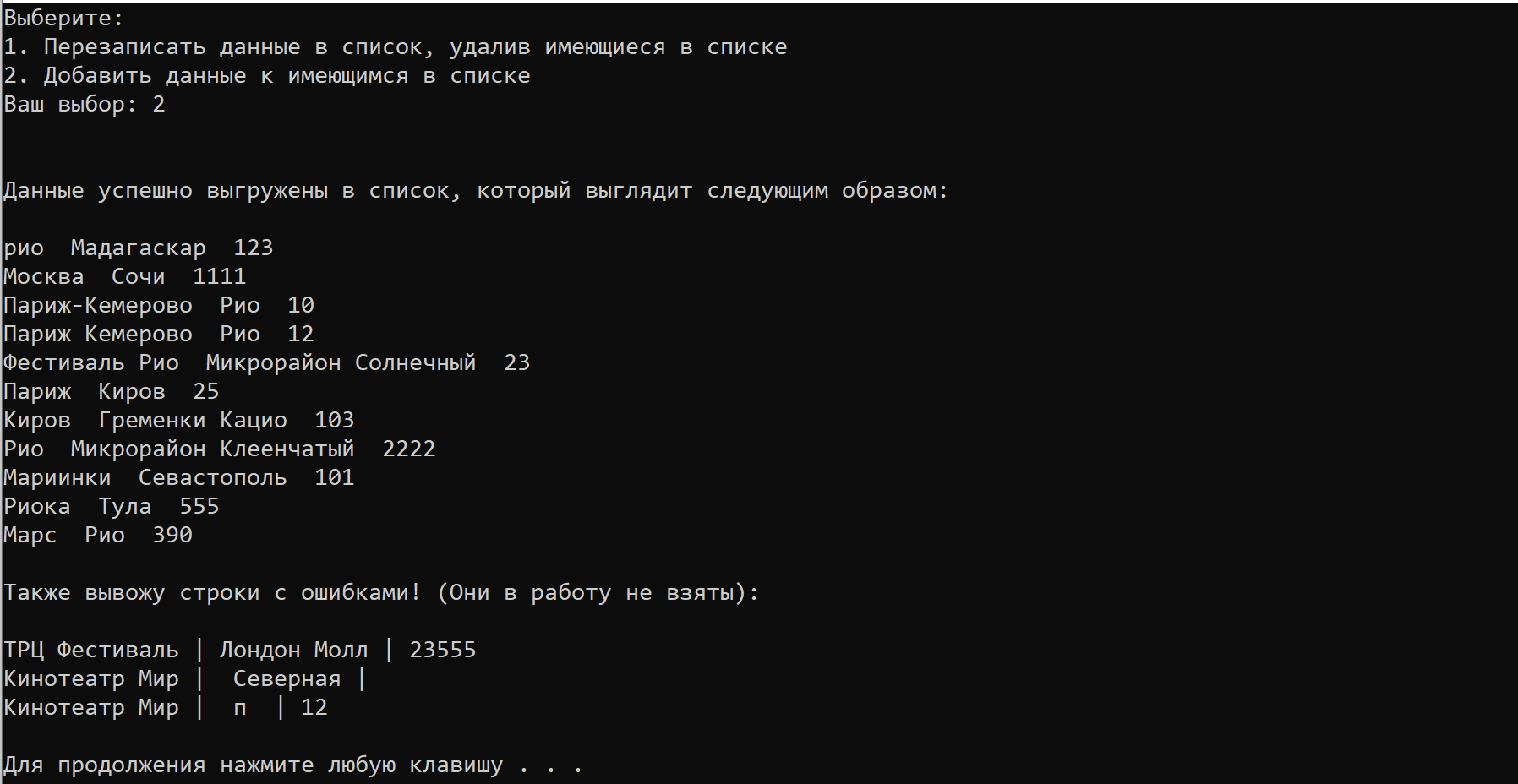


Рисунок 33 – Загрузка данных из файла при добавлении данных

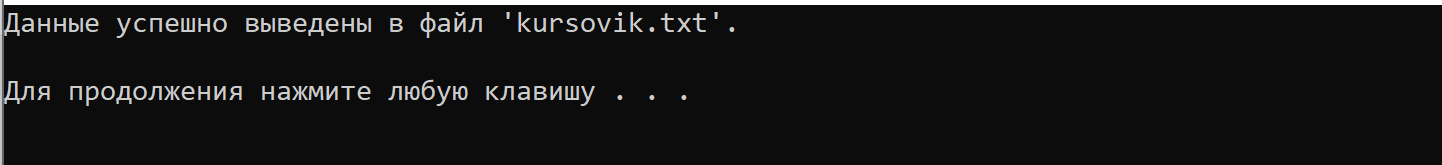


Рисунок 34 – Сохранение данных в файл

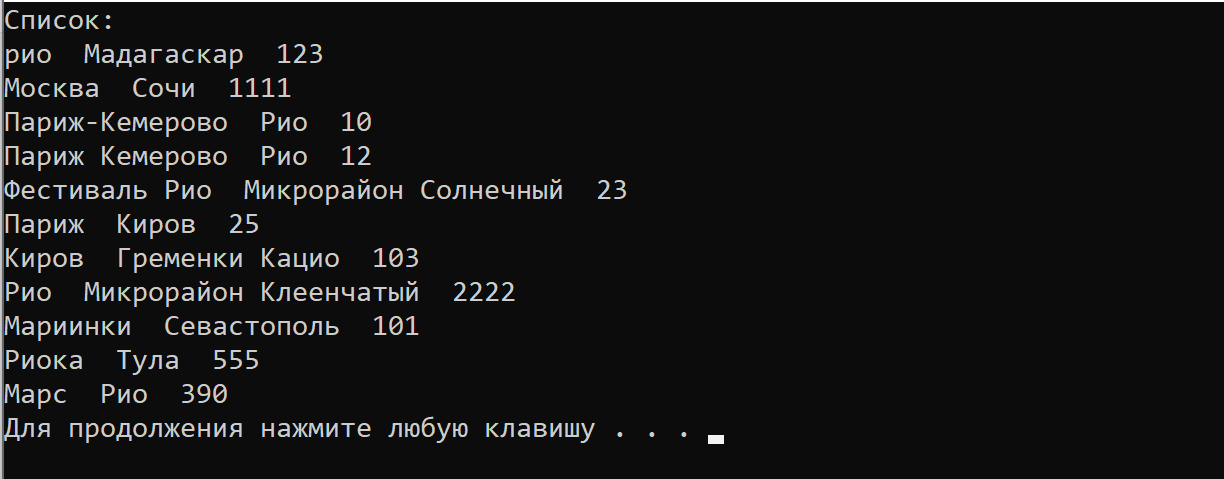


Рисунок 35 – Вывод списка на экран

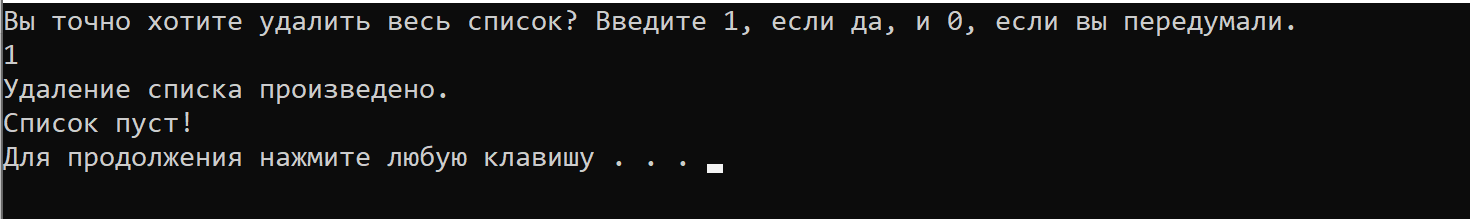


Рисунок 36 – Удаление списка, подтверждение получено

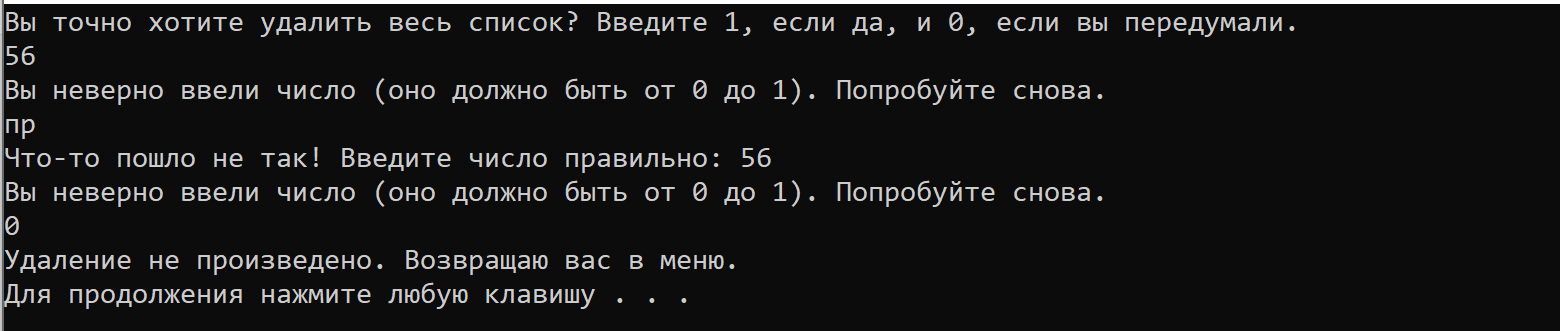


Рисунок 37 – Удаление списка с проверкой ввода выбора, подтверждение не получено

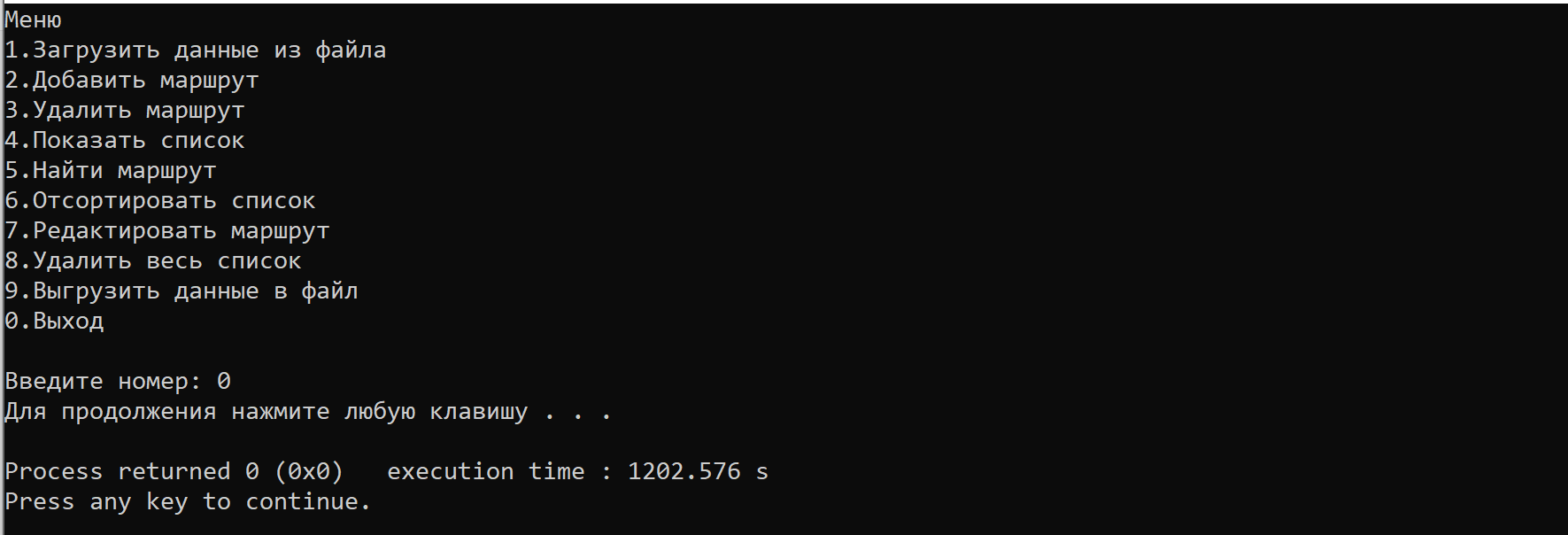


Рисунок 38 – Выход из программы

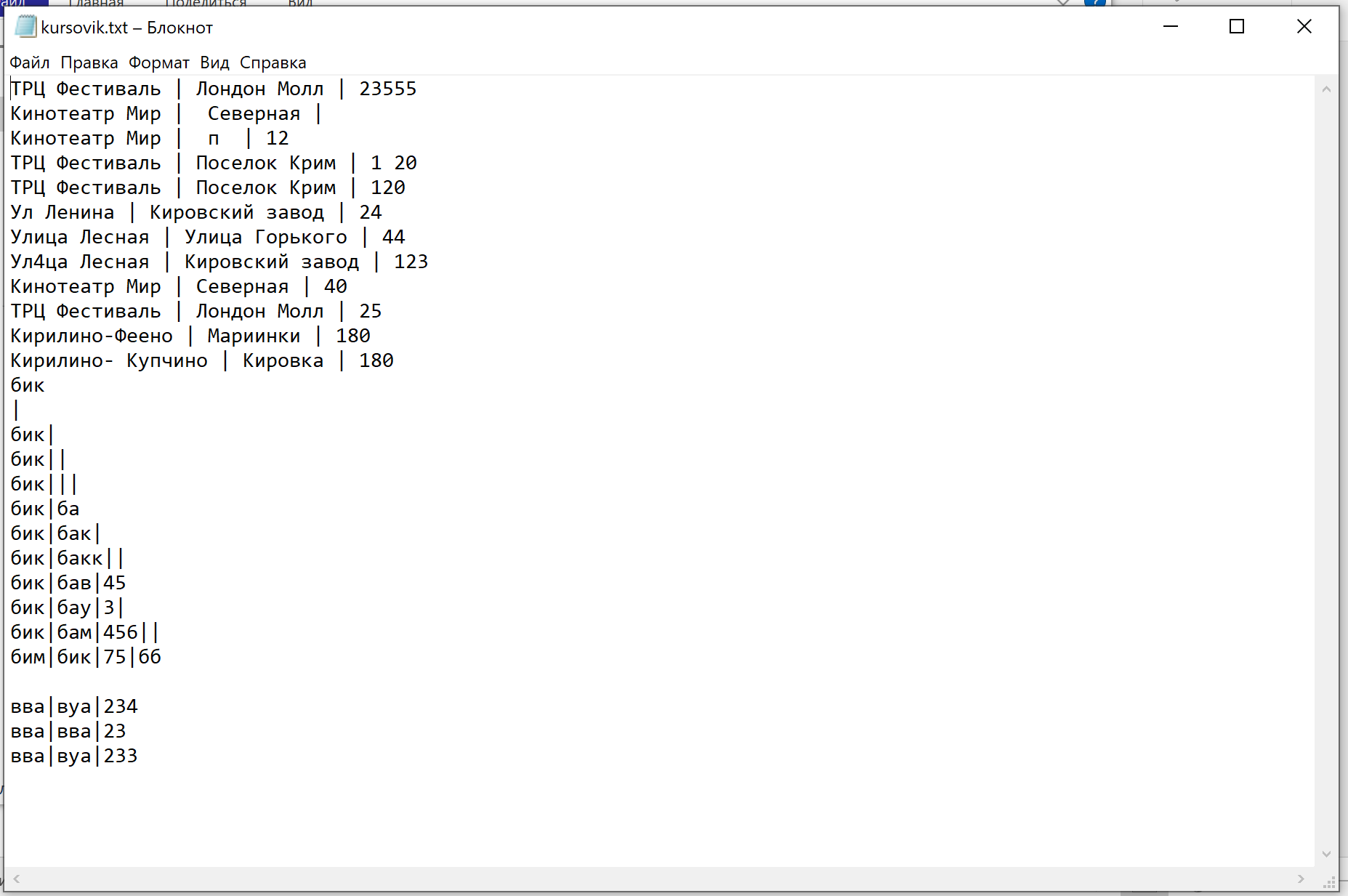


Рисунок 39 – Пример файла для загрузки

ЗАКЛЮЧЕНИЕ

Получены навыки работы со списками. Реализована программа работы со списком маршрутов. Успешно выполненная работа, с комплексным применением знаний прошлого семестра, демонстрирует освоение курса: Основы программирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* + 1. [https://stackoverflow.com](https://stackoverflow.com/)
    2. [https://www.cyberforum.ru](https://www.cyberforum.ru/)
    3. [https://habr.com](https://habr.com/)
    4. [https://ru.cppreference.com](https://ru.cppreference.com/w/)
    5. [https://learn.microsoft.com](https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170)

Приложение А. Текст программы

/\*#define \_CRTDBG\_MAP\_ALLOC

#include <stdlib.h>

#include <crtdbg.h>

#ifdef \_DEBUG

#ifndef DBG\_NEW

#define DBG\_NEW new ( \_NORMAL\_BLOCK , \_\_FILE\_\_ , \_\_LINE\_\_ )

#define newDBG\_NEW

#endif

#endif\*/

#include <iostream>

#include <string>

#include <fstream>

#include <algorithm>

#include <windows.h>

**using** **namespace** std;

**struct** MARSH

{

string startinPoint; //Начальный пункт маршрута

string destination; //Конечный пункт маршрута

string routeNumber; //Номер маршрута

};

**struct** List

{

MARSH marsh;

List \*next;

};

**bool** **readingFile**(List \*\*, List \*\*);

**int** **writingFile**(List \*);

**bool** **chekAndConversionStr**(string &); //Проверка корректности нач и кон пунктов маршрута

**bool** **chekAndConversionNumb**(string &); //Проверка корректности номера маршрута

**void** **addEnd**(List \*\*startList, MARSH temporaryRoute);

**void** **addStart**(List \*\*startList, MARSH temporaryRoute);

**bool** **addIndex**(List \*\*startList, MARSH temporaryRoute, **const** **int** index);

**void** **deleteList**(List \*\*); //Удаление списка полностью

**bool** **deleteEl**(List \*\*, MARSH, **const** **int**); //Удаление элемента (int - статус удаления)

**bool** **deleteEl**(List \*\*, string); //Удаление элемента по уникальному ключу - номеру маршрута (используется в deleteEl предыдущем)

**void** **print\_List**(List \*);

**bool** **findEl**(List \*, List \*\*, string);

**bool** **findSubstring**(**const** string, **const** string); //Функция для поиска подстроки в строке (используется в findEl)

**void** **redact**(List \*, string, string, **int**); //Конкретно меняет то, что было, на то, что надо

MARSH **findMarsh**(List \*, string); //Поиск маршрута по его номеру (номер проверен на совпадения)

**void** **printMarsh**(MARSH); //Вывод конкретного маршрута на экран

**void** **redactEl**(List \*, string); //Функция редактирования

**bool** **isMoreThan** (string, string); //Больше ли str1 чем str2

**bool** **isEquals** (string, string); //Сравнивает на равенство 2 строки, лояльна к регистру

**bool** **findEl**(MARSH, List \*); //Поиск, есть ли подобные маршруты в списке (проверка на уникальность)

**int** **clickIsOk** (); //Проверка на правильность введённого int

**void** **sortList**(List \*\*, **int**);

**int** **main**()

{

//\_CrtSetDbgFlag( \_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF);

setlocale(LC\_ALL, "rus");

SetConsoleCP(**1251**);

SetConsoleOutputCP(**1251**);

List \*startList = NULL, \*startErrorList = NULL;

List \*startFindList = NULL; //Список для функции поиска

**bool** flag = false;

**int** click = **0**, click2 = **0**, index;

MARSH temporaryRoute;

string firstOrEnd; //Для функции поиска

**bool** stop = false;

**while** (!stop)

{

system("cls");

cout << "Меню" << endl;

cout << "1.Загрузить данные из файла" << endl;

cout << "2.Добавить маршрут" << endl;

cout << "3.Удалить маршрут" << endl;

cout << "4.Показать список" << endl;

cout << "5.Найти маршрут" << endl;

cout << "6.Отсортировать список" << endl;

cout << "7.Редактировать маршрут" << endl;

cout << "8.Удалить весь список" << endl;

cout << "9.Выгрузить данные в файл" << endl;

cout << "0.Выход" << endl << endl;

cout << "Введите номер: ";

click = clickIsOk ();

**while**(click > **9** || click < **0**)

{

cout << "Вы неверно ввели число (оно должно быть от 1 до 9). Введите еще раз: ";

click = clickIsOk ();

}

**switch** (click)

{

**case** **1**:

system("cls");

click2 = **0**;

cout << "Выберите:" << endl;

cout << "1. Перезаписать данные в список, удалив имеющиеся в списке" << endl;

cout << "2. Добавить данные к имеющимся в списке" << endl;

cout << "Ваш выбор: ";

click2 = clickIsOk ();

**if** (click2 > **2** || click2 < **1**)

{

cout << "Вы неверно ввели число (оно должно быть от 1 до 2). Возвращайтесь в меню." << endl;

**break**;

}

**if** (click2 == **1**)

deleteList(&startList);

**if** (!readingFile(&startList, &startErrorList))

{

cout << "Ошибка! Файл не открыт!" << endl;

**return** -**1**;

}

**if** (startList == NULL && startErrorList == NULL) //Если в файле не было ни одной строки (некорретные в том числе)

{

cout << endl << "Файл пуст!" << endl << endl;

**break**;

}

cout << endl << endl << "Данные успешно выгружены в список, который выглядит следующим образом:" << endl << endl;

print\_List(startList);

**if** (startErrorList != NULL)

{

cout << endl << "Также вывожу строки с ошибками! (Они в работу не взяты):" << endl << endl;

print\_List(startErrorList);

deleteList(&startErrorList);

}

**else**

cout << endl << "Ошибочных строк в файле нет" << endl << endl;

**break**;

**case** **2**:

system("cls");

click2 = **0**;

cout << "Выберите:" << endl;

cout << "1. Добавить маршрут в конец списка" << endl;

cout << "2. Добавить маршрут в начало списка" << endl;

cout << "3. Добавить маршрут в список по индексу" << endl;

cout << "Ваш выбор: ";

click2 = clickIsOk ();

**if** (click2 > **3** || click2 < **1**)

{

cout << "Вы неверно ввели число (оно должно быть от 1 до 3). Возвращайтесь в меню." << endl;

**break**;

}

**while** (!flag)

{

cout << "Введите начальный пункт маршрута: ";

getline(cin, temporaryRoute.startinPoint);

**while** (!chekAndConversionStr(temporaryRoute.startinPoint))

{

cout << "Данные введены некорректно. Попробуйте еще раз: ";

getline(cin, temporaryRoute.startinPoint);

}

cout << "Введите конечный пункт маршрута: ";

getline(cin, temporaryRoute.destination);

**while** (!chekAndConversionStr(temporaryRoute.destination))

{

cout << "Данные введены некорректно. Попробуйте еще раз: ";

getline(cin, temporaryRoute.destination);

}

cout << "Введите номер маршрута: ";

getline(cin, temporaryRoute.routeNumber);

**while** (!chekAndConversionNumb(temporaryRoute.routeNumber))

{

cout << "Данные введены некорректно. Попробуйте еще раз: ";

getline(cin, temporaryRoute.routeNumber);

}

**if** (findEl(temporaryRoute, startList)) //Если введённый маршрут не уникален

cout << endl << "Маршрут не уникален. Попробуйте еще раз." << endl << endl;

**else**

flag = true; //Можно идти дальше, маршрут прошел проверки

}

flag = false;

**if** (click2 == **1**)

{

addEnd(&startList, temporaryRoute);

cout << endl << "Данные введены корректно. Маршрут добавлен в список:" << endl;

}

**if** (click2 == **2**)

{

addStart(&startList, temporaryRoute);

cout << endl << "Данные введены корректно. Маршрут добавлен в список:" << endl;

}

**if** (click2 == **3**)

{

cout << "Введите индекс, куда хотите вставить новый маршрут (от 1): ";

index = clickIsOk();

**while**(!addIndex(&startList, temporaryRoute, index))

{

cout << "Невозможно вставить элемент по данному индексу. Попробуйте еще раз. Индекс: ";

index = clickIsOk();

}

cout << endl << "Данные введены корректно. Маршрут добавлен в список:" << endl;

}

cout << endl;

print\_List(startList);

**break**;

**case** **3**:

system("cls");

click2 = **0**;

cout << "Список выглядит следующим образом:" << endl;

print\_List(startList);

cout << endl << "Выберите:" << endl;

cout << "1. Удалить все маршруты по названию начального пункта" << endl;

cout << "2. Удалить все маршруты по названию конечного пункта" << endl;

cout << "3. Удалить маршрут по номеру маршрута" << endl << endl;

cout << "Ваш выбор: ";

click2 = clickIsOk ();

**while** (click2 > **3** || click2 < **1**)

{

cout << "Вы неверно ввели число (оно должно быть от 1 до 3). Попробуйте снова." << endl;

click2 = clickIsOk ();

}

**if** (click2 == **1**)

{

cout << "Введите начальный пункт маршрута: " ;

getline(cin, temporaryRoute.startinPoint);

**while** (!chekAndConversionStr(temporaryRoute.startinPoint))

{

cout << "Данные введены некорректно. Попробуйте снова: ";

getline(cin, temporaryRoute.startinPoint);

}

**if** (!deleteEl(&startList, temporaryRoute, **1**))

{

cout << "Элементы, удовлетворяющие запросу, не найдены. Возвращайтесь в меню." << endl;

**break**;

}

}

**if** (click2 == **2**)

{

cout << "Введите конечный пункт маршрута: ";

getline(cin, temporaryRoute.destination);

**while**(!chekAndConversionStr(temporaryRoute.destination))

{

cout << "Данные введены некорректно. Попробуйте снова: ";

getline(cin, temporaryRoute.destination);

}

**if** (!deleteEl(&startList, temporaryRoute, **2**))

{

cout << "Элементы, удовлетворяющие запросу, не найдены. Возвращайтесь в меню." << endl;

**break**;

}

}

**if** (click2 == **3**)

{

cout << "Введите номер маршрута: ";

getline(cin, temporaryRoute.routeNumber);

**while**(!chekAndConversionNumb(temporaryRoute.routeNumber))

{

cout << "Данные введены некорректно. Попробуйте снова: ";

getline(cin, temporaryRoute.routeNumber);

}

**if** (!deleteEl(&startList, temporaryRoute, **3**))

{

cout << "Элементы, удовлетворяющие запросу, не найдены. Возвращайтесь в меню." << endl;

**break**;

}

}

cout << endl << "Удаление произведено. Список выглядит следующим образом:" << endl;

print\_List(startList);

**break**;

**case** **4**:

system("cls");

cout << "Список:" << endl;

print\_List(startList);

**break**;

**case** **5**:

system("cls");

cout << "Введите название пункта, по которому нужно осуществить поиск: ";

**while** (!flag)

{

getline(cin, firstOrEnd);

**if** (!chekAndConversionStr(firstOrEnd)) //Если строка введена корректно

cout << "Данные введены некорректно. Попробуйте еще раз: ";

**else**

{

**if** (!findEl(startList, &startFindList, firstOrEnd)) //Если ни одного элемента не найдено

{

**if** (startList == NULL)

cout << "Ничего не найдено, потому что список пуст!" << endl;

**else**

cout << "Маршрутов, удовлетворяющих условиям поиска, в списке нет." << endl;

flag = true;

}

**else**

{

cout << endl << "Найденные маршруты:" << endl << endl;

print\_List(startFindList);

flag = true;

}

}

}

flag = false;

deleteList(&startFindList);

**break**;

**case** **6**:

system("cls");

sortList(&startList, **1**);

**break**;

**case** **7**:

system("cls");

cout << "Введите номер маршрута, который хотите отредактировать: " << endl;

getline(cin, temporaryRoute.routeNumber);

**while** (!chekAndConversionNumb(temporaryRoute.routeNumber)) //Если строка введена корректно

{

cout << "Номер маршрута введен некорректно. Попробуйте еще раз." << endl;

getline(cin, temporaryRoute.routeNumber);

}

**if** (findEl(temporaryRoute, startList)) //Удостоверились, что элемент с таким номером есть в нашем списке

redactEl(startList, temporaryRoute.routeNumber);

**else**

{

cout << "Маршрут с таким номером не найден. Возвращайтесь в меню." << endl;

**break**;

}

cout << endl << "Список выглядит следующим образом:" << endl;

print\_List(startList);

**break**;

**case** **8**:

system("cls");

**if** (startList == NULL)

{

cout << "Список пуст, нет данных для удаления!" << endl;

**break**;

}

cout << "Вы точно хотите удалить весь список? Введите 1, если да, и 0, если вы передумали." << endl;

click2 = clickIsOk ();

**while** (click2 > **1** || click2 < **0**)

{

cout << "Вы неверно ввели число (оно должно быть от 0 до 1). Попробуйте снова." << endl;

click2 = clickIsOk ();

}

**if** (click2)

{

cout << "Удаление списка произведено." << endl;

deleteList(&startList);

print\_List(startList);

}

**else**

cout << "Удаление не произведено. Возвращаю вас в меню." << endl;

**break**;

**case** **9**:

system("cls");

**if** (!writingFile(startList))

{

cout << "Ошибка! Файл для записи не открыт!" << endl;

deleteList(&startList);

**return** -**1**;

}

**if** (writingFile(startList) == **1**)

cout << "Данные успешно выведены в файл 'kursovik.txt'." << endl << endl;

**else**

cout << "Ваш список пуст, в файл ничего не передано, предыдущие данные затерты!" << endl << endl;

**break**;

**case** **0**:

stop = true;

deleteList(&startList);

**break**;

}

system("pause");

}

/\*\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDOUT);

\_CrtDumpMemoryLeaks();

system("pause");\*/

**return** **0**;

}

**bool** **readingFile**(List \*\*startList, List \*\*startErrorList)

{

fstream ioputFile;

ioputFile.open("kursovik.txt", ios::in | ios::out);

**if** (!ioputFile.is\_open())

**return** false;

ioputFile.seekg(**0**, ios::end); //Перешли в конец файла

**if** (ioputFile.tellg() > **1**)

{

ioputFile.seekg(-**1**, ios::end);

**char** c;

ioputFile.get(c);

**if** (c != '\n')

ioputFile << '\n';

}

**else**

**return** true; //Файл прочитан, но список пуст - значит файл пуст

ioputFile.close();

ifstream inputFile;

inputFile.open("kursovik.txt", ifstream::binary);

**if** (!inputFile.is\_open())

**return** false;

**while** (!inputFile.eof())

{

**int** cursor;

string s;

cursor = inputFile.tellg();

getline(inputFile, s, '\n');

**if** ((**int**)s[s.length()-**1**] == **13**)

s.erase(s.length()-**1**, **1**); //Удаляем каретку

MARSH temporaryErrorRoute;

temporaryErrorRoute.startinPoint = s;

**int** amount = **0**;

**for** (**size\_t** i = **0**; i < s.length(); i++)

{

**if** (s[i] == '|')

amount++;

}

**if** (amount != **2**)

addEnd(startErrorList, temporaryErrorRoute); //Записываем данную строку в список ошибочных строк

**else**

{

inputFile.seekg(cursor, ios::beg); //Перемещаем курсор на начало строки

MARSH temporaryRoute; //Временный маршрут, чтобы не портить оригинал

getline(inputFile, temporaryRoute.startinPoint, '|');

getline(inputFile, temporaryRoute.destination, '|');

getline(inputFile, temporaryRoute.routeNumber);

**if** (chekAndConversionStr(temporaryRoute.startinPoint) && chekAndConversionStr(temporaryRoute.destination) && chekAndConversionNumb(temporaryRoute.routeNumber))

{

//Если введённый маршрут не уникален (мы смогли его найти в списке) или нач и кон пункты идентичны

**if** (findEl(temporaryRoute, \*startList) || isEquals(temporaryRoute.startinPoint, temporaryRoute.destination))

addEnd(startErrorList, temporaryErrorRoute); //Записываем данную строку в список ошибочных строк

**else**

addEnd(startList, temporaryRoute);

}

**else**

addEnd(startErrorList, temporaryErrorRoute);

}

}

inputFile.close();

List \*stErList = \*startErrorList;

**if** (stErList != NULL)

**if** (stErList->next == NULL) //Если в списке с ошибками всего 1 строка

**if** (stErList->marsh.startinPoint.length() == **0**) //И она пустая

deleteList(startErrorList); //Если из всех строк всего одна ошибочная и то пустая, то считаем, что ош строк нет

**return** true; //Файл успешно прочитан

}

**int** **writingFile**(List \*stList)

{

ofstream outputFile;

outputFile.open("kursovik.txt");

**if** (!outputFile.is\_open())

**return** **0**;

**if** (stList == NULL)

**return** **2**;

**while** (stList != NULL)

{

outputFile << stList->marsh.startinPoint << " | " << stList->marsh.destination << " | " << stList->marsh.routeNumber;

**if** (stList->next != NULL)

outputFile << endl;

stList = stList->next;

}

outputFile.close();

**return** **1**;

}

**bool** **chekAndConversionStr**(string &str) //Корректность дефиса, присутствие ё

{

string s = "";

**bool** flag = true; //Сейчас идут пробелы

**for** (**size\_t** i = **0**; i < str.length(); i++) //Убираем пробелы и проверяем корректность символов

{

**if** (str[i] == ' ')

{

**if** (flag == false)

s += ' ';

flag = true;

}

**else**

{

flag = false; //Началась или продолжается запись слова

**if** (((**int**)str[i] < -**64** || (**int**)str[i] > -**1**) && (**int**)str[i] != (**int**)'-' && str[i] != 'ё')

**return** false;

**else**

{

**if** (str[i] == 'ё')

s += 'е';

**else**

s += str[i];

}

}

}

**if** (s[s.length()-**1**] == ' ')

s.erase(s.length()-**1**, **1**);

**int** k = -**1**; //Длина текущего слова

**int** k1 = **0**; //Кол-во '-'

**for** (**size\_t** i = **0**; i < s.length(); i++) //Проверка на длину слова (больше 2)

{

**if** (s[i] == ' ')

{

**if** (k < **3**)

**return** false;

k = **0**;

}

**else**

{

**if** (k == -**1**)

k = **0**;

**if** ((**int**)s[i] >= -**64** || (**int**)s[i] <= -**1**)

k++;

**if** (s[i] == '-')

{

k1++;

**if** (k1 > **1** || s[i-**1**] == ' ' || s[i+**1**] == ' ')

**return** false;

}

}

}

**if** (k < **3**)

**return** false;

str = s;

**return** true;

}

**bool** **chekAndConversionNumb**(string &str) //Корректное число - от 0 до 9999, также недопустимо: 040

{

string s = "";

**bool** flag1 = false; //Запись номера не начата

**bool** flag2 = false; //Запись номера не закончена

**for** (**size\_t** i = **0**; i < str.length(); i++)

{

**if** (str[i] == ' ')

{

**if** (flag1) //Если запись номера уже начиналась, значит она закончена

flag2 = true;

}

**else**

{

**if** (!flag1) //Если запись номера не начиналась, то она начинается сейчас

{flag1 = true;}

**if** (flag1) //Если запись номера начиналась

{

**if** (!flag2) //И не закончилась

{

**if** (((**int**)str[i] < **48** || (**int**)str[i] > **58**) && (**int**)str[i] != **13**)

**return** false;

**else**

{

**if** ((**int**)str[i] != **13**)

s += str[i];

}

}

**else** //и закончилась (а значит какие-то еще символы появились

**return** false;

}

}

}

**if** (s.length() > **0**)

{

**if** (s.length() > **4** || s[**0**] == '0')

**return** false;

}

**else**

**return** false;

str = s;

**return** true;

}

**void** **addEnd**(List \*\*startList, MARSH temporaryRoute)

{

List \*stList = \*startList;

**if** (\*startList == NULL)

{

\*startList = **new** List;

List \*stList = \*startList;

stList->marsh.startinPoint = temporaryRoute.startinPoint;

stList->marsh.destination = temporaryRoute.destination;

stList->marsh.routeNumber = temporaryRoute.routeNumber;

stList->next = NULL;

}

**else**

{

**while** (stList->next != NULL)

stList = stList->next;

stList->next = **new** List;

stList->next->marsh.startinPoint = temporaryRoute.startinPoint;

stList->next->marsh.destination = temporaryRoute.destination;

stList->next->marsh.routeNumber = temporaryRoute.routeNumber;

stList->next->next = NULL;

}

}

**void** **addStart**(List \*\*startList, MARSH temporaryRoute)

{

**if** (\*startList == NULL)

addEnd(startList, temporaryRoute);

**else**

{

List \*oldStList = \*startList; //Запомнили адрес "бывшего" первого элемента

List \*newStList = **new** List; //Создали новый элемент

//Заполняем значения нового элемента:

newStList->marsh.startinPoint = temporaryRoute.startinPoint;

newStList->marsh.destination = temporaryRoute.destination;

newStList->marsh.routeNumber = temporaryRoute.routeNumber;

newStList->next = oldStList;

//Перемещаем указатель на начало списка на новый первый элемент:

\*startList = newStList;

}

}

**bool** **addIndex**(List \*\*startList, MARSH temporaryRoute, **int** index)

{

**bool** flag = false; //Если false - значит элемент не был записан

**if** (index == **1**)

{

addStart(startList, temporaryRoute);

flag = true;

}

**else**

{

List \*stList = \*startList;

**if** (stList == **0**)

flag = false;

**else**

{

**int** k = **1**;

**while** (stList->next != NULL)

{

**if** (k+**1** == index)

{

List \*oldNext = stList->next; //Запомнили элемент, следующий за тем, который мы добавим

//Выделяем память и записываем добавленный элемент

stList->next = **new** List;

stList->next->marsh.startinPoint = temporaryRoute.startinPoint;

stList->next->marsh.destination = temporaryRoute.destination;

stList->next->marsh.routeNumber = temporaryRoute.routeNumber;

stList->next->next = oldNext; //Не потеряли связь с конечной частью списка

flag = true; //Элемент записан

}

stList = stList->next; //Двигаемся по списку

k++;

}

**if** (k + **1** == index)

{

flag = true; //Элемент записан

addEnd(startList, temporaryRoute);

}

}

}

**return** flag; //Записан элемент, или же индекс неверно указан

}

**int** **lengthList**(List \*\*startList)

{

**int** length = **0**;

List \*stList = \*startList;

**if** (stList == NULL)

**return** **0**;

**while** (stList != NULL)

{

length++;

stList = stList->next;

}

**return** length;

}

**void** **sortList**(List \*\*startList, **int** status)

{

**int** m = lengthList(startList) - **1**, length = lengthList(startList);

**if** (m > **0**)

{

cout << "Список до сортировки:" << endl;

print\_List(\*startList);

**while** (m > **0**)

{

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

{

**if** ((i + m) < length)

{

string a = "", b = "";

//this запоминает адрес нужного для замены элемента, before - адрес элемента до нужного для замены, after - после заменяемого

List \*before1, \*before2, \*this1, \*after1, \*this2, \*after2, \*stList2 = \*startList, \*stList3 = \*startList;

List \*before, \*after;

**int** j = **0**;

**while** (stList2->next != NULL)

{

**if** (m != **1**)

{

**if** (j == **0** && i == **0**) //Значит обрабатываемый элемент - первый элемент в списке

{

before1 = NULL; //Просто пометим, чтобы знать, что нужно переставить ук на начало списка

this1 = stList2; //stList2 сейчас указывает на голову списка

after1 = stList2->next; //Указатель на 2 эл списка

a = stList2->marsh.startinPoint;

}

**else**

{

**if** (j + **1** == i) //Обрабатываемый элемент - от второго в списке

{

before1 = stList2;

this1 = stList2->next;

after1 = stList2->next->next;

a = stList2->next->marsh.startinPoint;

}

}

**if** (j + **1** == i + m)

{

before2 = stList2;

this2 = stList2->next;

after2 = stList2->next->next;

b = stList2->next->marsh.startinPoint;

}

}

**else**

{

**if** (j == **0** && i == **0**) //Значит обрабатываемый элемент - первый элемент в списке

{

this1 = stList2; //stList2 сейчас указывает на голову списка

this2 = stList2->next; //Указатель на 2 эл списка

before = NULL; //Просто пометим, чтобы знать, что нужно переставить ук на начало списка

after = this1->next->next;

a = stList2->marsh.startinPoint;

b = stList2->next->marsh.startinPoint;

}

**else**

{

**if** (j + **1** == i) //Обрабатываемый элемент - от второго в списке

{

this1 = stList2->next;

this2 = stList2->next->next;

before = stList2;

after = stList2->next->next->next;

a = stList2->next->marsh.startinPoint;

b = stList2->next->next->marsh.startinPoint;

}

}

}

stList2 = stList2->next;

j++;

}

**if** (isMoreThan (a, b) && !isEquals (a, b) && a != "" && b != "")

{

**if** (m != **1**)

{

List \*thisTemp = this2;

**if** (before1 == NULL) //Здесь нужно поменять голову списка

{

before2->next = this1; //Переставили новый элемент

before2->next->next = after2; //Заполнили его next предыдущим after

\*startList = thisTemp; //Переставили новый элемент

stList3 = \*startList; //Доп указатель, временный, указывает на новый первый элемент

stList3->next = after1; //Заполнили его next предыдущим after

}

**else**

{

before2->next = this1;

before2->next->next = after2;

before1->next = thisTemp;

before1->next->next = after1;

}

}

**else**

**if** (before == NULL) //Здесь нужно поменять голову списка

{

\*startList = this2;

stList3 = \*startList;

stList3->next = this1;

stList3->next->next = after;

}

**else**

{

before->next = this2;

before->next->next = this1;

before->next->next->next = after;

}

}

}

**else**

**break**;

}

m -= **1**;

}

cout << endl <<"Сортировка выполнена. Отсортированный список:" << endl;

print\_List(\*startList);

}

**else**

cout << endl << "Список негоден для сортировки. Возвращайтесь в меню." << endl;

}

**void** **print\_List**(List \*stList)

{

**if** (stList == NULL)

cout << "Список пуст!" << endl;

**while** (stList != NULL)

{

cout << stList->marsh.startinPoint << " " << stList->marsh.destination << " " << stList->marsh.routeNumber << endl;

stList = stList->next;

}

}

**void** **deleteList**(List \*\*startList)//Удалить весь список

{

List \*stList = \*startList;

**if** (\*startList != NULL)

{

List \*newStList = stList->next;

**while** (newStList != NULL)

{

**delete** stList;

\*startList = newStList;

stList = \*startList;

newStList = stList->next;

}

**delete** \*startList;

\*startList = NULL;

}

}

**bool** **deleteEl**(List \*\*startList, string routeNumb)

{

List \*stList = \*startList;

**if** (stList->marsh.routeNumber == routeNumb) //Значит самый первый элемент - искомый

{

\*startList = stList->next;

**delete** stList;

**return** true;

}

**else**

{

**while** (stList->next != NULL) //Пока в списке есть хотя бы два элемента

{

**if** (stList->next->next != NULL) //Проверка на то, что второй по счету элемент от \*stList не последний

{

**if** (stList->next->marsh.routeNumber == routeNumb) //Рассматриваем элемент, который находится за тем, на который указывает stList

{

List \*delEl = stList->next; //Запоминаем элемент, который нужно удалить

stList->next = delEl->next; //Переставили указатель

stList = delEl->next; //Перешли на след элемент

**delete** delEl;

**return** true;

}

}

**else** //В списке осталось 2 элемента, рассматриваем последний

{

**if** (stList->next->marsh.routeNumber == routeNumb) //Рассматриваем элемент, который находится за тем, на который указывает stList

{

List \*delEl = stList->next; //Запоминаем элемент, который нужно удалить

stList->next = NULL; //Переставили указатель

**delete** delEl;

**return** true;

}

}

stList = stList->next;

}

}

**return** false;

}

**bool** **deleteEl**(List \*\*startList, MARSH temporaryRoute, **const** **int** status) //Проверить статус до входа

{

**bool** flag = false; //Был ли найден и удален элемент

List \*stList = \*startList;

**if** (stList == NULL)

**return** false; //Список пуст

**else**

{

**if** (status == **1**)//Значит нужно удалить все элементы по начальному пункту

{

**while** (stList != NULL)

{

**if** (isEquals(stList->marsh.startinPoint, temporaryRoute.startinPoint))

{

**if** (!deleteEl(startList, stList->marsh.routeNumber))

flag = false;

**else**

flag = true;

}

stList = stList->next;

}

}

**if** (status == **2**)//Значит нужно удалить все элементы по конечному пункту

{

**while** (stList != NULL)

{

**if** (isEquals(stList->marsh.destination, temporaryRoute.destination))

{

**if** (!deleteEl(startList, stList->marsh.routeNumber))

flag = false;

**else**

flag = true;

}

stList = stList->next;

}

}

**if** (status == **3**)//Значит нужно удалить элемент по номеру, введённому с клавиатуры

{

**if** (!deleteEl(startList, temporaryRoute.routeNumber))

flag = false;

**else**

flag = true;

}

}

**return** flag;

}

**bool** **findEl**(List \*stList, List \*\*startFindList, string firstOrEnd) //Задание на поиск по названию пунктов маршрута

{

**bool** flag = false;

**if** (stList == NULL)

**return** false; //Список пуст

**else**

{

**while** (stList != NULL)

{

**if** (findSubstring(stList->marsh.startinPoint, firstOrEnd) || findSubstring(stList->marsh.destination, firstOrEnd))

{

addEnd(startFindList, stList->marsh);

flag = true;

}

stList = stList->next;

}

}

**return** flag;

}

**bool** **findEl**(MARSH temporaryRoute, List \*stList) //Поиск, есть ли подобные маршруты в списке (проверка на уникальность)

{

**if** (stList == NULL)

**return** false; //Список пуст, подобных маршрутов нет - ОК

**else**

{

**while** (stList != NULL)

{

//Проверка на совпадение

**if** ((isEquals(stList->marsh.startinPoint, temporaryRoute.startinPoint) && isEquals(stList->marsh.destination, temporaryRoute.destination)) || stList->marsh.routeNumber == temporaryRoute.routeNumber)

**return** true;

stList = stList->next;

}

}

**return** false;

}

**bool** **findSubstring**(**const** string str, **const** string subStr)

{

string temporaryStr;

**if** (str.length() < subStr.length())

**return** false;

**for** (**size\_t** i = **0**; i < (str.length() - subStr.length() + **1**); i++)

{

**size\_t** temp = i;

**for** (**size\_t** j = **0**; j < subStr.length(); j++)

{

temporaryStr += str[temp];

temp++;

}

**if** (isEquals(temporaryStr, subStr))

**return** true;

temporaryStr = "";

}

**return** false;

}

**void** **redact**(List \*stList, string routeNumb, string replaceStr, **int** status)

{

**while** (stList != NULL)

{

//Проверка на совпадение

**if** (stList->marsh.routeNumber == routeNumb)

{

**if** (status == **1**)

{

stList->marsh.startinPoint = replaceStr;

}

**if** (status == **2**)

{

stList->marsh.destination = replaceStr;

}

**if** (status == **3**)

{

stList->marsh.routeNumber = replaceStr;

}

**break**;

}

stList = stList->next;

}

}

MARSH **findMarsh**(List \*stList, string routeNumb) //Поиск маршрута по его номеру (номер проверен на совпадения)

{

MARSH temporaryRoute;

**while** (stList != NULL)

{

//Выводим выбранный маршрут, чтобы показать, как он выглядит

**if** (stList->marsh.routeNumber == routeNumb)

{

temporaryRoute.startinPoint = stList->marsh.startinPoint;

temporaryRoute.destination = stList->marsh.destination;

temporaryRoute.routeNumber = stList->marsh.routeNumber;

**return** temporaryRoute;

}

stList = stList->next;

}

**return** temporaryRoute;

}

**void** **printMarsh**(MARSH temporaryRoute)//Вывод конкретного маршрута на экран

{

cout << "Маршрут выглядит следующим образом:" << endl;

cout << temporaryRoute.startinPoint << " " << temporaryRoute.destination << " " << temporaryRoute.routeNumber << endl;

}

**void** **redactEl**(List \*stList, string routeNumb) //Функция редактирования

{

**bool** flag1 = false; //Маячок о том, есть ли ошибка в веденных данных

MARSH temporaryRouteOrig;

temporaryRouteOrig.routeNumber = routeNumb;

**int** click2 = **0**;

**while** (click2 != **4**)

{

MARSH temporaryRouteTemp;

temporaryRouteOrig = findMarsh(stList, temporaryRouteOrig.routeNumber);

printMarsh(temporaryRouteOrig);

cout << "Выберите:" << endl;

cout << "1. Редактировать начальный пункт" << endl;

cout << "2. Редактировать конечный пункт" << endl;

cout << "3. Редактировать номер" << endl;

cout << "4. Закончить редактирование" << endl << endl;

cout << "Ваш выбор: ";

click2 = clickIsOk ();

**while** (click2 > **4** || click2 < **1**)

{

cout << "Вы неверно ввели число (оно должно быть от 1 до 4). Попробуйте снова: ";

click2 = clickIsOk ();

}

**if** (click2 == **1**)

{

cout << "Введите начальный пункт маршрута, на который хотите заменить действующий: ";

**while** (!flag1)

{

getline(cin, temporaryRouteTemp.startinPoint);

**if** (!chekAndConversionStr(temporaryRouteTemp.startinPoint))

cout << "Данные введены некорректно. Попробуйте снова: ";

**else**

**if** (isEquals (temporaryRouteOrig.startinPoint, temporaryRouteTemp.startinPoint))

cout << "Вы ввели то же, что и было. Редактирование не произведено. Попробуйте снова: ";

**else**

**if** (isEquals (temporaryRouteOrig.destination, temporaryRouteTemp.startinPoint))

cout << "Редактирование не произведено, ведь иначе начальный и конечный пункты будут совпадать. Попробуйте снова: ";

**else**

{

string number = temporaryRouteOrig.routeNumber; //Запоминаем текущий номер маршрута

temporaryRouteOrig.startinPoint = temporaryRouteTemp.startinPoint;

temporaryRouteOrig.routeNumber = -**1**; //Делаем это для проверки на уникальность

**if** (findEl(temporaryRouteOrig, stList)) //Если после изменений наш маршрут станет неуникальным, то не позволяем редактирование

{

temporaryRouteOrig.routeNumber = number; //Возвращаем корректный номер

cout << "Данное редактирование невозможно, маршрут станет неуникальным. Попробуйте снова: ";

}

**else**

{

temporaryRouteOrig.routeNumber = number; //Возвращаем корректный номер

redact(stList, temporaryRouteOrig.routeNumber, temporaryRouteTemp.startinPoint, **1**);

flag1 = true;

}

}

}

}

**if** (click2 == **2**)

{

cout << "Введите конечный пункт маршрута, на который хотите заменить действующий: ";

**while** (!flag1)

{

getline(cin, temporaryRouteTemp.destination);

**if** (!chekAndConversionStr(temporaryRouteTemp.destination))

cout << "Данные введены некорректно. Попробуйте снова: ";

**else**

**if** (isEquals (temporaryRouteOrig.destination, temporaryRouteTemp.destination))

cout << "Вы ввели то же, что и было. Редактирование не произведено. Попробуйте снова: ";

**else**

**if** (isEquals (temporaryRouteOrig.startinPoint, temporaryRouteTemp.destination))

cout << "Редактирование не произведено, ведь иначе начальный и конечный пункты будут совпадать. Попробуйте снова: ";

**else**

{

string number = temporaryRouteOrig.routeNumber; //Запоминаем текущий номер маршрута

temporaryRouteOrig.destination = temporaryRouteTemp.destination;

temporaryRouteOrig.routeNumber = -**1**; //Делаем это для проверки на уникальность

**if** (findEl(temporaryRouteOrig, stList)) //Если после изменений наш маршрут станет неуникальным, то не позволяем редактирование

{

temporaryRouteOrig.routeNumber = number; //Возвращаем корректный номер

cout << "Данное редактирование невозможно, маршрут станет неуникальным. Попробуйте снова: ";

}

**else**

{

temporaryRouteOrig.routeNumber = number; //Возвращаем корректный номер

redact(stList, temporaryRouteOrig.routeNumber, temporaryRouteTemp.destination, **2**);

flag1 = true;

}

}

}

}

**if** (click2 == **3**)

{

cout << "Введите номер маршрута, на который хотите заменить действующий: ";

**while** (!flag1)

{

getline(cin, temporaryRouteTemp.routeNumber);

**if** (!chekAndConversionNumb(temporaryRouteTemp.routeNumber))

cout << "Данные введены некорректно. Попробуйте снова: ";

**else**

**if** (temporaryRouteOrig.routeNumber == temporaryRouteTemp.routeNumber)

cout << "Вы ввели то же, что и было. Редактирование не произведено. Попробуйте снова: ";

**else**

{

string number = temporaryRouteOrig.routeNumber; //Запоминаем текущий номер маршрута

string stPoint = temporaryRouteOrig.startinPoint; //Запомнили нач пункт, потому что мы его будем менять

temporaryRouteOrig.startinPoint = "-1"; //Для проверки на уникальность

temporaryRouteOrig.routeNumber = temporaryRouteTemp.routeNumber;

**if** (findEl(temporaryRouteOrig, stList)) //Если после изменений наш маршрут станет неуникальным, то не позволяем редактирование

{

//Если редактирование невозможно, то возвращаем номер маршрута

temporaryRouteOrig.routeNumber = number;

cout << "Данное редактирование невозможно, маршрут станет неуникальным. Попробуйте снова: ";

}

**else**

{

redact(stList, number, temporaryRouteTemp.routeNumber, **3**);

flag1 = true;

}

}

}

}

**if** (click2 == **4**)

**break**;

**if** (flag1)

{

cout << endl << "Редактирование произведено." << endl;

temporaryRouteOrig = findMarsh(stList, temporaryRouteOrig.routeNumber);

printMarsh(temporaryRouteOrig);

}

flag1 = false;

cout << endl;

system("pause");

system("cls");

}

cout << "Редактирование завершено." << endl;

}

**bool** **isMoreThan** (string str1, string str2) //Больше ли str1 чем str2

{

**size\_t** len = min(str1.length(), str2.length());

**char** c1, c2;

**for** (**size\_t** i = **0**; i < len; i++)

{

**if** ((**int**)str1[i] >= -**64** && (**int**)str1[i] <= -**33**)

c1 = str1[i] + **32**;

**else**

c1 = str1[i];

**if** ((**int**)str2[i] >= -**64** && (**int**)str2[i] <= -**33**)

c2 = str2[i] + **32**;

**else**

c2 = str2[i];

**if** (c1 != c2)

{

**if** (c2 == '-')

{

**if** (c1 == ' ')

**return** true;

**else**

**return** false;

}

**else**

{

**if** (c1 == '-')

{

**if** (c2 == ' ')

**return** false;

**else**

**return** true;

}

**else**

{

**if** (c1 > c2)

**return** true;

**else**

**return** false;

}

}

}

}

//Если мы оказались тут, то одна из строк является подстрокой другой строки

**if** (str1.length() > str2.length())

**return** true;

**else**

**return** false;

}

**bool** **isEquals** (string str1, string str2) //Функция сравнения строк на равенство, лояльная к регистру

{

**if** (str1.length() != str2.length())

**return** false;

**for** (**size\_t** i = **0**; i < str1.length(); i++)

{

**if** ((**int**)str1[i] >= -**64** && (**int**)str1[i] <= -**33**)

str1[i] = str1[i] + **32**;

**if** ((**int**)str2[i] >= -**64** && (**int**)str2[i] <= -**33**)

str2[i] = str2[i] + **32**;

}

**if** (str1 == str2)

**return** true;

**else**

**return** false;

}

**int** **clickIsOk** ()

{

**int** result;

cin >> result;

**while** (cin.fail())

{

cin.clear();

cin.ignore(INT\_MAX, '\n');

cout << "Что-то пошло не так! Введите число правильно: ";

cin >> result;

}

cin.ignore(INT\_MAX, '\n');

**return** (result);

}