

**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»

Пояснительная записка к курсовой работе

по дисциплине: «Программирование»

на тему: «Система управления базой данных»

Выполнил студент группы М3О-211Б-21: Богомольский Виталий

Проверили: Дмитриева Елена Анатольевна и Ивашенцев Иван Владимирович

Москва 2022 г.

# Содержание

# Задание

**Задания на курсовую работу по курсу «Программирование»**

Сдать на проверку и защитить работу необходимо до 24 декабря 2022 года

Разработать программу на языке программирования С++ и обеспечить:

* выбор ввода исходных данных из файла и с клавиатуры,
* выбор вывода результатов на экран или в файл.

В пояснительной записке необходимо привести:

* постановку задачи,
* входные данные,
* структурные схемы алгоритмов,
* описание интерфейса программы,
* текст программы с комментариями,
* результаты проверки функционирования программы,
* выходные данные,
* выводы.

**Задание на курсовую работу**

Разработать систему управления базой данных на языке С++.

Тематики базы данных выбираются согласно варианту.

Для хранения данных использовать текстовый файл.

Разработать структуры данных для хранения данных и функции для их обработки.

Разработать функции для ввода данных, выборки значения по заданному условию, вывода данных на экран, удаления, записи новых данных, сортировки по определенному пользователем значению. Возможно наличие нескольких узлов списков с одинаковыми значениями полей. Обеспечить выдачу запросов по любым полям данных. Реализовать систему меню.

**Схема организации данных**

1. Линейные односвязные списки
2. Кольцевые односвязные списки
3. Линейные двусвязные списки
4. Кольцевые двусвязные списки

**Требования, предъявляемые к оформлению текста пояснительной записки**

***Нумерация работы***, начиная с титульного листа, выполняется арабскими цифрами в нижней части страницы (правый угол). Титульный лист считается первым, но не нумеруется.

Пояснительная записка:

* титульный лист;
* содержание;
* задание;
* краткое описание процесса разработки (схемы алгоритмов, описание функций);
* код программы( модулей);
* результаты работы;
* выводы.

**Содержание -** обеспечить автоматическое формирование содержания с применением встроенных стилей заголовков Word. Стиль текста - **Times New Roman,** 12, красная строка, выравнивание – по ширине, межстрочный интервал 1,5.

**Код** должен быть дополнен подробными комментариями. Все **результаты** (каждая картинка, «скрин» экрана) должны сопровождаться подробными описаниями.

**Образец** титульного листа представлен.

# Краткое описание процесса разработки (схемы алгоритмов, описание функций)

## Описание интерфейса программы

Программа представляет собой консольное приложение. При запуске программы предлагается произвести предварительную настройку программы: произвести выбор типа используемого списка (линейный односвязный или линейный двухсвязный), а также ввести стартовые значения названия вокзала и тарифа для проезда, которые могут быть позднее изменены в меню.

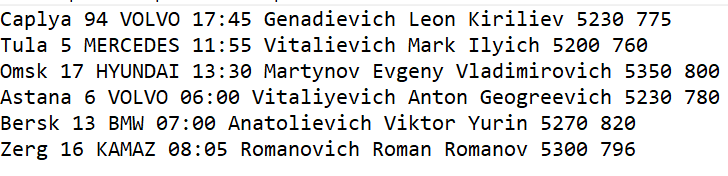
После предварительной настройки пользователь видит перед собой меню, которое представляет из себя: название автовокзала, краткая информация о нем (тариф введенный на маршрутах) и 14 опций для выполнения. Пользователь выбирает одну из 14 опций и далее переносится в подменю для каждой из опций, уже в нем реализуется логика каждой конкретной опции.

Выход из меню и завершение работы программы производятся вызовом 14-ой опции. Все опции, которые требовалось реализовать в задании, представлены в меню. Для работы с программой используется 3 файла: входных данных, хранения БД и выходных данных. Названия файлов пользователем не вводятся, они указаны в коде программы (output.txt, bus\_station.txt, input.txt).

Работа с интерфейсом программы осуществляется либо последовательным вводом каких-либо значений, либо через файл, если данная опция его использует.

## Входные данные

Входные данные (помимо настройки программы в консоли) представляют собой информацию об автобусах автобусной станции, включающую такие данные как номер его рейса, время отправления и другое. Вводятся входные данные либо через консоль, либо из файла. Пример файла входных данных (направление, номер рейса, марка, время отправления, ФИО водителя, число проданных билетов за неделю, прогнозируемое число пассажиров за сегодня):



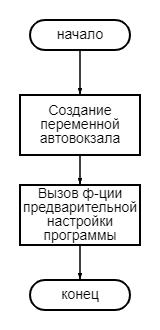
## Структурные схемы алгоритмов

Отметим, что сама программа разбита на несколько файлов. Особыми функциями являются main, функция предварительной настройки программы и функция меню - эти три функции обязательно последовательно вызываются в начале программы. Далее вызов функций будет зависеть от выбора пользователя. В меню представлены 14 опций, которые по своему смыслу разбиты на 5 групп: связанные с добавлением или изменением данных, выводом данных, удалением данных, сортировкой данных, закрытием программы. Каждой из 14 опций меню соответствует своя функция, которая отвечает за взаимодействие с пользователем и реализацию только самой поверхностной части логики функции. При этом более сложным из этих 14-функций приходится вызывать дополнительные функции, реализованные также в отдельных файлах. Пример: опции меню номер 1 соответствует функция; она предложит пользователю вернуться в меню, если он случайно вызвал эту функцию и если есть риск потери данных, а затем вызовет уже более “глубокие” функции: если установлен тип списка “односвязный линейный” она вызовет функцию ввода данных для него, в противном случае, для “двухсвязного”. Таким образом, все 14 опций меню привязаны к как бы “принимающим решения” функциям, а уже те вызывают “исполнительные, содержащие основную логику”.

Программа запускается, вызываются функции: main -> console\_programm\_start -> console\_menu.

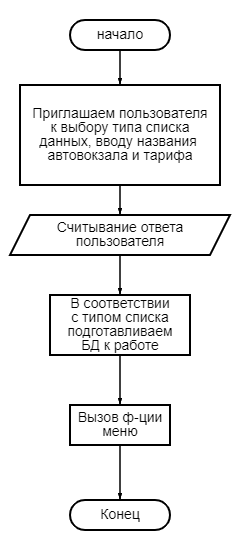
#### Функция int main():

Основная программа. “Конец” возвращает 0 (если где-либо в коде не было кода ошибки).



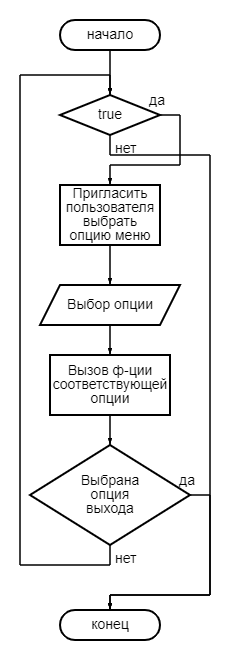
#### Функция void console\_programm\_start():

Предварительная настройка программы. Ничего не возвращает (тип void).



#### Функция void console\_menu ():

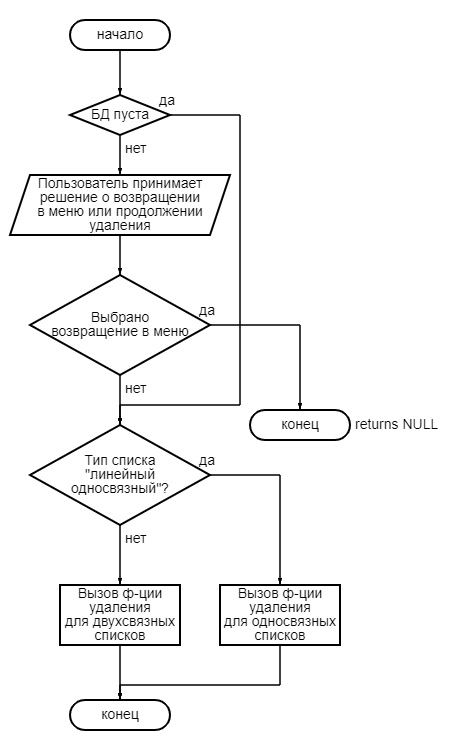
Функция меню. Ничего не возвращает (тип void). Как видно, выход из цикла меню происходит только при выборе пользователем соответствующей опции.



Были приведены три основных функции программы. Приведем еще несколько функций в пример.

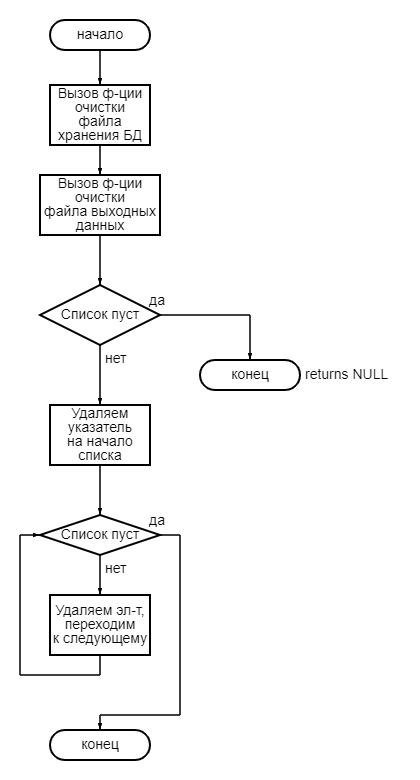
#### void console\_menu\_option\_delete\_all\_data():

Функция, соответствующая 11-ой опции меню: удаление всех данных. Ничего не возвращает (тип void).



#### void delete\_all\_data():

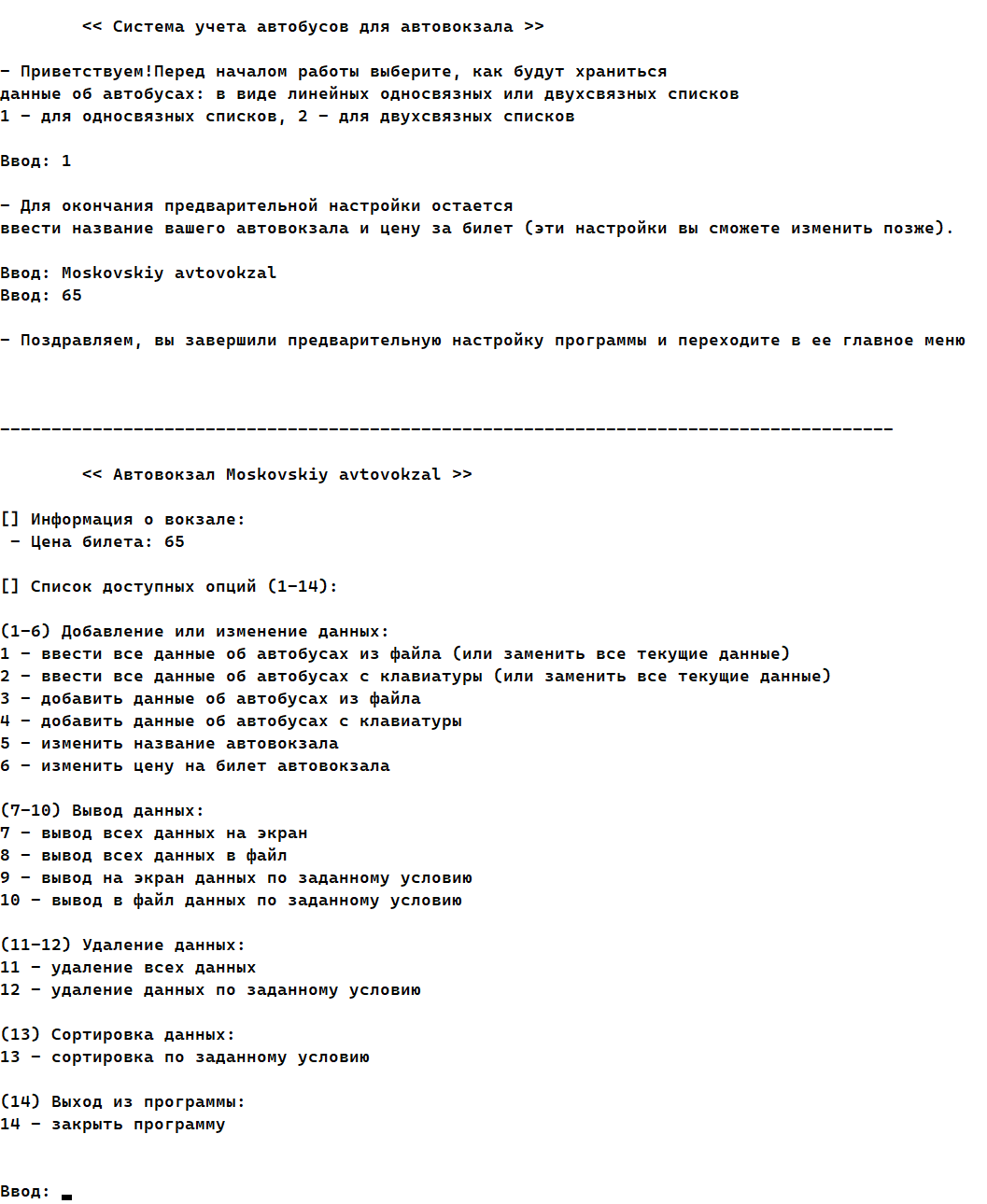
Удаляет все данные: из списка, файла хранения данных и файла выходных данных. В программе используется перегрузка: эта функция в случае односвязных список и двухсвязных списков. Ничего не возвращает, тип void.



## 

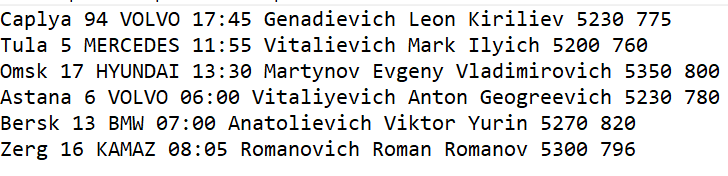
# Результаты проверки функционирования программы

Запуск программы. Выберем тип списка “линейный односвязный”, введем название автовокзала и выберем тариф.

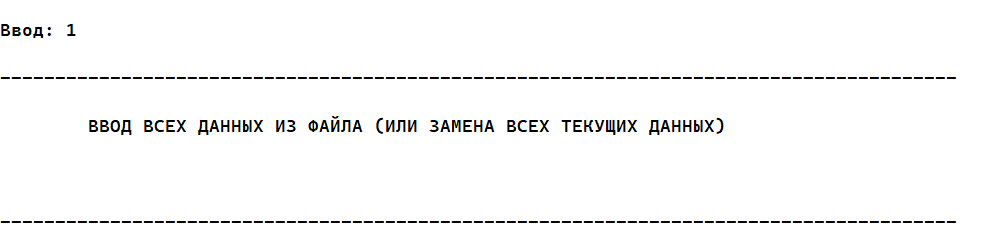


Как видно, успешно отобразилось меню. Нам представлены все опции на выбор, информация о вокзале отображается корректно.

Попробуем ввести данные из файла, в котором записано:



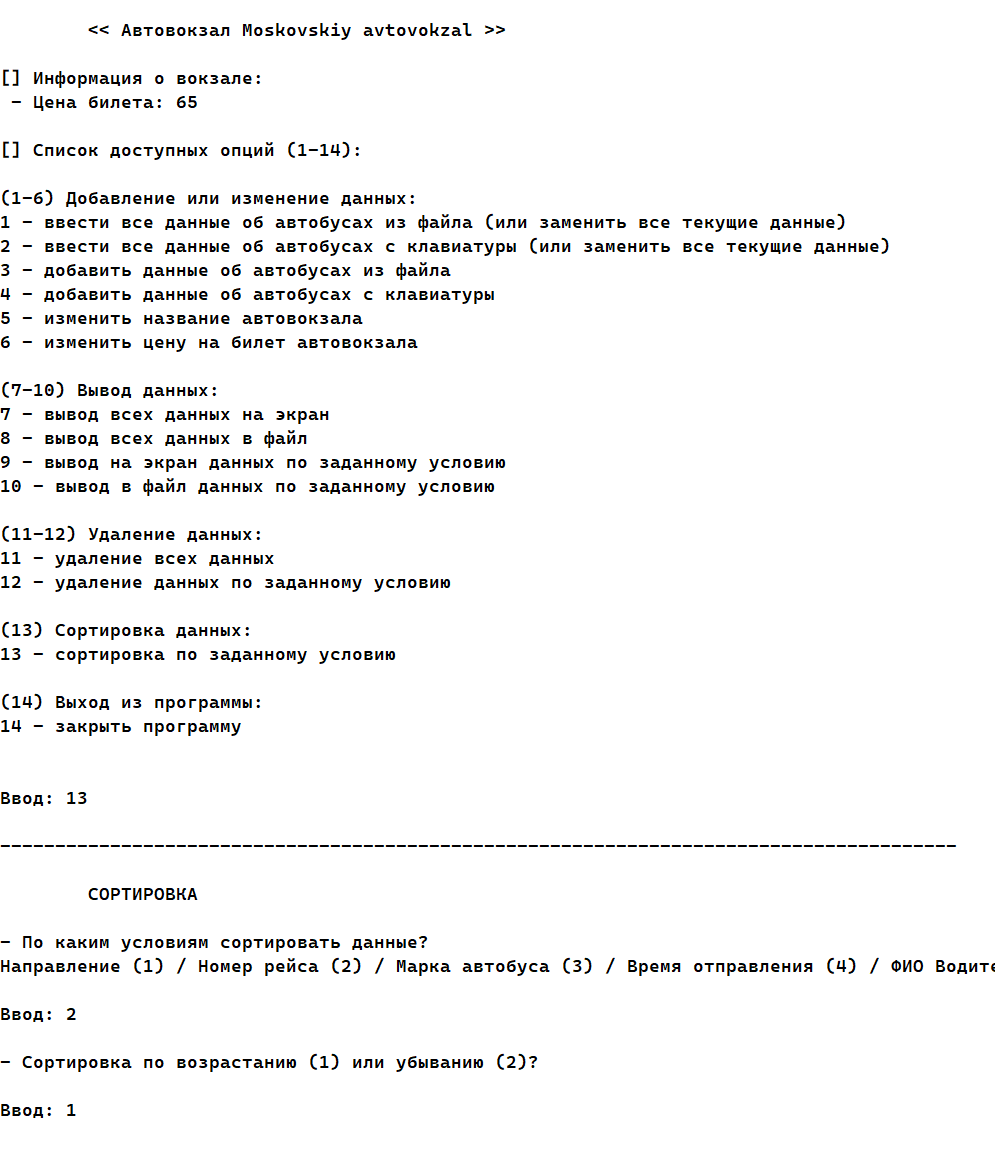
Выбираем опцию 1 (ввод из файла):



Выведем считанные данные с помощью опции 7 (вывод в консоль):

Довольно широкий список данных, который даже трудно уместить на вертикальноориентированной бумаге, однако все данные отобразились корректно, как и ожидалось.

Попробуем отсортировать их по номеру рейса в порядке возрастания, вызовем 13-ю опцию меню:



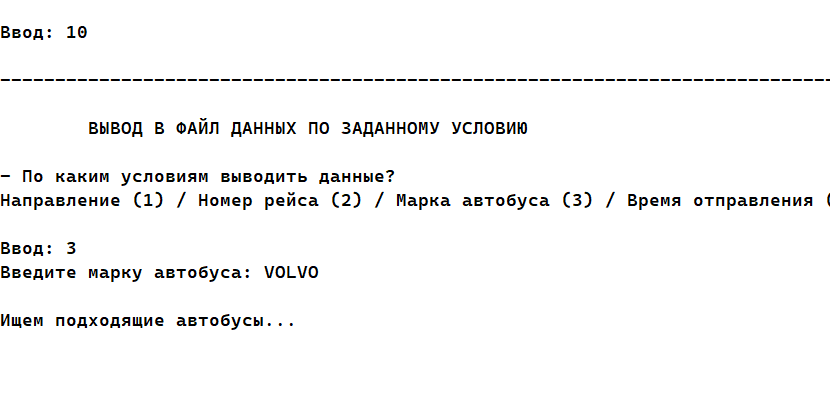
Проверим результаты сортировки, выведя список автобусов в консоль:



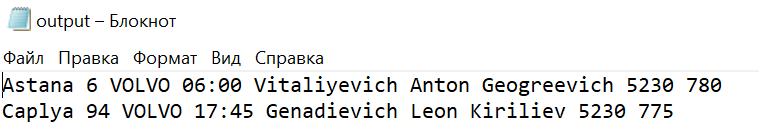
Отлично, функция сортировки сработала так, как надо.

Все те же опции при их вызове неотличимо так же работают для двухсвязных списков, разница только в реализации в самом коде, пользователь разницы не замечает.

Наконец, попробуем вывести данные в файл. Выведем в файл автобусы марки VOLVO. Вызовем опцию 10, вывод в файл по заданному значению:

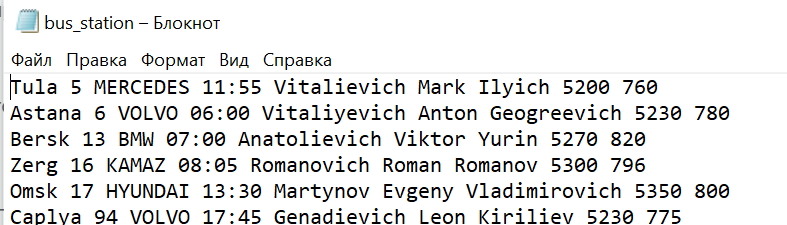


В результате файл выходных данных содержит:



Отлично!

Кстати, все это время БД хранилась не только в программе, но и в файле хранения данных:



После сортировки произошла синхронизация, и они совпадают с теми данными, что сейчас хранятся в программе.

Закрывается программа вызовом 14-ой опции. Файлы хранения БД и выходных данных очищаются.

Для двухсвязного списка приводить скриншоты программы не имеет смысла, так как визуально, стоит повториться, пользователь не видит никакой разницы при работе программы для разных видов списков.

# Код программы (модулей)

Приведем код основных трех модулей (main, функция преднастройки, функция меню) и еще нескольких функций. По приведенному коду выше были представлены блок-схемы.

#### Файл main.cpp:

| // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // ----------------------------------- Курсовая работа ----------------------------------------  // --------------------------------------------------------------------------------------------  // --------------------------------Файл основной программы-------------------------------------  // --------------------------------------------------------------------------------------------  // Подключение стандартных библиотек:  #include <iostream>  // Подключение стандартного пространста имен:  using namespace std;  // Подключение других файлов программы:  #include "structures\_definition.h" // Файл с определением введенных структур (включает, в частности, определение структуры автобусной станции bus\_station)  #include "start\_and\_menu\_functions.h" // Файл с заголовками функций начала программы и меню  // Основная программа:  int main() {  // Поддержка русского языка в консоли:  setlocale(LC\_ALL, "Russian");  // Переменая для хранения базы данных "Автобусная станция":  bus\_station\* bs = new bus\_station;  // Начнем исполнение программы с вызова соответствующей функции:  console\_programm\_start(bs);  } // Конец блока "Основная программа" |
| --- |

#### Файл start\_and\_menu\_functions.cpp:

| // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // ----------------------------------- Курсовая работа ----------------------------------------  // --------------------------------------------------------------------------------------------  // ----------------------Файл определение функций начала программы и меню----------------------  // --------------------------------------------------------------------------------------------  // Подключение стандартных библиотек:  #include <string>  #include <iostream>  // Подключение стандартного пространста имен:  using namespace std;  // Подключение других файлов программы:  #include "structures\_definition.h" // Файл с определением вводимых структур (включает, в частности,  // определение структуры автобусной станции bus\_station, структуру списка автобусов и т.д.)  #include "additional\_functions.h" // Файл с заголовками дополнительных функций (таких как "является ли строка числом от 1 до n)  #include "menu\_options\_all\_groups\_together.h" // Файл с заголовками функций всех пяти групп опций меню (группа опций меню "Ввод", "Вывод" и  // т.д. - всего 5 таких групп, и в них в сумме 14 функций - "Ввод из консоли", "Ввод из файла" и т.д.).  // То есть опции меню делятся на группы по смыслу, а в каждой группе есть по несколько конкретных функций.  // Объявление функций:  // Старт программы:  void console\_menu(bus\_station\* bs);  // Меню:  void console\_programm\_start(bus\_station\* bs);  // Определение функций:  // Старт программы:  void console\_programm\_start(bus\_station\* bs) {  // Объявление переменных:  string list\_type; // Тип списка, который будет использоваться в БД (линейный односвязный или линейный двухсвязный)  string bus\_station\_name; // Название данной автобусной станции  int bus\_station\_ticket\_price; // Цена билета, установленная для проезда на всех автобусах данной станции  // Приветственный заголовок:  cout << "\n\t<< Система учета автобусов для автовокзала >>\n\n- Приветствуем!";  // Приглашаем пользователя выбрать тип списка:  cout << "Перед началом работы выберите, как будут храниться\n"  "данные об автобусах: в виде линейных односвязных или двухсвязных списков\n"  "1 - для односвязных списков, 2 - для двухсвязных списков\n\n";  // Получаем от пользователя ответ о типе списка:  list\_type = get\_user\_input\_with\_conditions(0, "is\_1\_or\_2"); // Получение ввода пользователя с проверкой на корректность: была ли введена единица или двойка  // Приглашаем пользователя выбрать название для автовокзала и цену на билет  // для всех автобусов данной станции (позднее эти параметры можно будет изменить):  cout << "\n- Для окончания предварительной настройки остается\n"  "ввести название вашего автовокзала и цену за билет "  "(эти настройки вы сможете изменить позже).\n\n"  "Ввод: ";  // Получаем от пользователя ответ с названием автовокзала:  cin.ignore(); // Очистка буфера ввода  getline(cin, bus\_station\_name); // Стандартный метод ввода, позволяющий считать строку из нескольких слов в одну переменную типа string    // Получаем от пользователя ответ с ценой за билет:  bus\_station\_ticket\_price = stoi(get\_user\_input\_with\_conditions(0, "natural\_number")); // Получение ввода пользователя с проверкой на корректность: было ли введено натуральное число  // Если пользователь выбрал тип списка "линейный односвязный список", то создаем его, а двухсвязный список не используем:  if (list\_type == "1") {  bs->bus\_station\_name = bus\_station\_name; // Записываем в БД название автовокзала  bs->bus\_station\_ticket\_price = bus\_station\_ticket\_price; // Записываем в БД цену за билет  bs->bus\_station\_type\_of\_list = '1'; // Записываем в БД тип списка - линейный односвязный список ('1' - односвязный, '2' - двухсвязный)  bs->bus\_station\_buses\_linear\_singly\_linked\_list = new buses\_linear\_singly\_linked\_list; // Выделяем в БД память под односвязный список  bs->bus\_station\_buses\_linear\_singly\_linked\_list->buses\_first\_bus = NULL; // В списке пока нет ни одного автобуса, устанавливаем NULL для первого автобуса в списке  bs->bus\_station\_buses\_linear\_doubly\_linked\_list = NULL; // Линейный двусвязный список не используется базой данных не используется  } // Конец блока "Если пользователь выбрал тип списка "линейный односвязный список", то создаем его, а двухсвязный список не используем"  // Если пользователь выбрал тип списка "линейный двухсвязный список", то создаем его, а односвязный список не используем:  if (list\_type == "2") {  bs->bus\_station\_name = bus\_station\_name; // Записываем в БД название автовокзала  bs->bus\_station\_ticket\_price = bus\_station\_ticket\_price; // Записываем в БД цену за билет  bs->bus\_station\_type\_of\_list = '2'; // Записываем в БД тип списка - линейный двухсвязный список ('1' - односвязный, '2' - двухсвязный)  bs->bus\_station\_buses\_linear\_doubly\_linked\_list = new buses\_linear\_doubly\_linked\_list; // Выделяем в БД память под двухсвязный список  bs->bus\_station\_buses\_linear\_doubly\_linked\_list->buses\_first\_bus = NULL; // В списке пока нет ни одного автобуса, устанавливаем NULL для первого автобуса в списке  bs->bus\_station\_buses\_linear\_doubly\_linked\_list->buses\_last\_bus = NULL; // В списке пока нет ни одного автобуса, устанавливаем NULL для последнего автобуса в списке  bs->bus\_station\_buses\_linear\_singly\_linked\_list = NULL; // Линейный односвязный список базой данных не используется  } // Конец блоков "Если пользователь выбрал тип списка "линейный двухсвязный список", то создаем его, а односвязный список не используем"  // Объявляем о завершении предварительной настройки программы:  cout << "\n- Поздравляем, вы завершили предварительную настройку программы и переходите в ее главное меню\n\n";    // Переводим пользователя в меню:  console\_menu(bs);  } // Конец блока "Старт программы"  // Меню:  void console\_menu(bus\_station\* bs) {  // Объявление переменных:  string menu\_option\_str = ""; // Выбранная пользователем опция меню (в типе string)  int menu\_options\_number = 14; // Число опций в меню  // Цикл меню (прерывается только вызовом опции выхода):  while (true) {  // Печатаем содержание меню:  cout << "\n\n---------------------------------------------------------------------------------------\n\n"  "\t<< Автовокзал " << bs->bus\_station\_name << " >>\n\n"  "[] Информация о вокзале:\n"  " - Цена билета: " << bs->bus\_station\_ticket\_price << "\n\n"  "[] Список доступных опций (1-14):\n"  "\n(1-6) Добавление или изменение данных:\n"  "1 - ввести все данные об автобусах из файла (или заменить все текущие данные)\n"  "2 - ввести все данные об автобусах с клавиатуры (или заменить все текущие данные)\n"  "3 - добавить данные об автобусах из файла\n"  "4 - добавить данные об автобусах с клавиатуры\n"  "5 - изменить название автовокзала\n"  "6 - изменить цену на билет автовокзала\n"  "\n(7-10) Вывод данных:\n"  "7 - вывод всех данных на экран\n"  "8 - вывод всех данных в файл\n"  "9 - вывод на экран данных по заданному условию\n"  "10 - вывод в файл данных по заданному условию\n"  "\n(11-12) Удаление данных:\n"  "11 - удаление всех данных\n"  "12 - удаление данных по заданному условию\n"  "\n(13) Сортировка данных:\n"  "13 - сортировка по заданному условию\n"  "\n(14) Выход из программы:\n"  "14 - закрыть программу\n\n\n";  // Получаем от пользователя ответ, содержащий выбор одной из опций меню:  menu\_option\_str = get\_user\_input\_with\_conditions(menu\_options\_number + 1, "lower\_then\_n"); // Получение ввода пользователя  // с проверкой на корректность:  // было ли введено натуральное  // число меньшее menu\_options\_number + 1 -  // то есть соответствует ли ввод  // пользователя одной из опций меню  // Визуально отделим меню консоли от далее вызываемой функции:  cout << "\n---------------------------------------------------------------------------------------\n\n";  // В соответствии с пользовательским вводом выберем опцию и вызовем ее функцию:  switch (stoi(menu\_option\_str)) { // stoi(menu\_option\_str) - это целочисленная запись  // содержимого строки menu\_option\_str, то есть номер опции меню,  // которую выбрал пользователь  case 1:  console\_menu\_option\_input\_data\_from\_file(bs); // Ввести все данные об автобусах из файла (или заменить все текущие данные)  break;  case 2:  console\_menu\_option\_input\_data\_from\_console(bs); // Ввести все данные об автобусах с клавиатуры (или заменить все текущие данные)  break;  case 3:  console\_menu\_option\_add\_data\_from\_file(bs); // Добавить данные об автобусах из файла  break;  case 4:  console\_menu\_option\_add\_data\_from\_console(bs); // Добавить данные об автобусах с клавиатуры  break;  case 5:  console\_menu\_option\_change\_bus\_station\_name(bs); // Изменить название автовокзала  break;  case 6:  console\_menu\_option\_change\_bus\_station\_ticket\_price(bs); // Изменить цену на билет автовокзала  break;  case 7:  console\_menu\_option\_output\_all\_data\_to\_console(bs); // Вывод всех данных на экран  break;  case 8:  console\_menu\_option\_output\_all\_data\_to\_file(bs); // Вывод всех данных в файл  break;  case 9:  console\_menu\_option\_output\_choosen\_data\_to\_console(bs); // Вывод на экран данных по заданному условию  break;  case 10:  console\_menu\_option\_output\_choosen\_data\_to\_file(bs); // Вывод в файл данных по заданному условию  break;  case 11:  console\_menu\_option\_delete\_all\_data(bs); // Удаление всех данных  break;  case 12:  console\_menu\_option\_delete\_choosen\_data(bs); // Удаление выбранных данных  break;  case 13:  console\_menu\_option\_sort\_data(bs); // Сортировка по заданному условию  break;  case 14:  console\_menu\_option\_exit\_programm(bs); // Закрыть программу  break;  default: // Пользовательский ввод проверен на то, чтобы хотя бы одна из опций выше обязательно была вызвана.  // Поэтому default никогда выполняться не должен.  cout << "Error in menu function"; // Сообщение об ошибке  exit(1); // Выход из программы с кодом ошибки 1  break;  } // Конец блока "В соответствии с пользовательским вводом выберем опцию и вызовем ее функцию"  } // Конец блока "Цикл меню"  } // Конец блока "Меню" |
| --- |

#### Файл menu\_options\_group\_of\_delete.cpp:

| // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // ----------------------------------- Курсовая работа ----------------------------------------  // --------------------------------------------------------------------------------------------  // -----------Файл определения функций опций меню из группы опций удаления данных--------------  // --------------------------------------------------------------------------------------------  // Подключение стандартных библиотек:  #include <iostream>  #include <fstream>  // Подключение стандартного пространста имен:  using namespace std;  // Подключение других файлов программы:  #include "structures\_definition.h" // Файл с определением вводимых структур (включает, в частности,  // определение структуры автобусной станции bus\_station, структуру списка автобусов и т.д.)  #include "additional\_functions.h" // Файл с заголовками дополнительных функций (таких как "является ли строка числом от 1 до n)  #include "group\_delete.h" // Файл с заголовками функций, содержащих главную часть алгоритма удаления данных  #include "group\_output.h" // Файл с заголовками функций, содержащих главную часть алгоритма вывода данных  #include "constants.h" // Файл с определением констант  // Объявление функций:  // Удаление всех данных:  void console\_menu\_option\_delete\_all\_data(bus\_station\* bs);  // Удаление выбранных данных:  void console\_menu\_option\_delete\_choosen\_data(bus\_station\* bs);  // Определение функций:  // Удаление всех данных:  void console\_menu\_option\_delete\_all\_data(bus\_station\* bs) {  cout << "\tУДАЛЕНИЕ ВСЕХ ДАННЫХ\n\n";  // Объявление переменных:  string user\_input; // Пользовательский ввод  // Если база данных не пуста (хранит данные об автобусах):  if (!is\_data\_base\_holding\_empty\_list(bs)) {  // Вывод на экран предупреждения о возможной потере данных с предложением вернуться в меню:  cout << "- Предупреждение: в базе данных уже хранятся данные об автобусах.\n"  "Вы уверены, что хотите их стереть (1) или выберите вернуться в меню (2)?\n\n";  // Получим от пользователя ответ, хочет ли он вернуться в меню:  user\_input = get\_user\_input\_with\_conditions(0, "is\_1\_or\_2"); // Получение от пользователя ответа с проверкой на корректность: введена ли единица или двойка  // Если пользователь выбрал вернуться в меню, то возвращаем его в меню:  if (user\_input == "2") {  return; // Прерываем выполнение функции типа void, вызванной из цикла меню. Возвращаемся в цикл меню.  } // Конец блока "Если пользователь выбрал вернуться в меню, то возвращаем его в меню"  } // Конец блока "Если база данных не пуста (хранит данные об автобусах)"  // Вызываем функцию удаления всех данных для линейного односвязного списка, если такой тип списка установлен в БД:  // Примечание: функция так же удалит и все данные из файла хранения данных и все данные из файла выходных данных  if (bs->bus\_station\_type\_of\_list == '1') delete\_all\_data(bs->bus\_station\_buses\_linear\_singly\_linked\_list);  // В случае двухсвязного списка, передаем в ту же функцию указатель на двухсвязный список, хранящийся в БД:  else delete\_all\_data(bs->bus\_station\_buses\_linear\_doubly\_linked\_list);  } // Конец блока "Удаление всех данных"  // Удаление выбранных данных:  void console\_menu\_option\_delete\_choosen\_data(bus\_station\* bs) {  cout << "\tУДАЛЕНИЕ ВЫБРАННЫХ ДАННЫХ\n\n";  // Объявление переменных:  int n = 7; // Число параметров удаления  int delete\_parameter; // Номер параметра удаления, выбранный пользователем (от 1 до 7)  string delete\_key; // Значение параметра удаления, по которому будут удаляться данные  cout << "- По каким условиям удалять данные?\n";  cout << "Направление (1)"  << " / " << "Номер рейса (2)"  << " / " << "Марка автобуса (3)"  << " / " << "Время отправления (4)"  << " / " << "ФИО Водителя (5)"  << " / " << "Число проданных билетов (6)"  << " / " << "Расчетное число пассажиров (7)\n\n";  // Считывание от пользователя номера параметра удаления с проверкой на корректность: было ли введено натуральное число:  delete\_parameter = stoi(get\_user\_input\_with\_conditions(n + 1, "lower\_then\_n"));  // В зависимости от номера параметра удаления, считываем параметр удаления  // (у разных параметров разный способ ввода)  switch (delete\_parameter) {  case 1:  cout << "Введите направление. ";  cin >> delete\_key;  break;  case 3:  cout << "Введите марку автобуса: ";  cin >> delete\_key;  break;  case 4:  cout << "Введите время отправления: ";  cin >> delete\_key;  break;  case 5:  cout << "Введите ФИО водителя: ";  cin.ignore();  getline(cin, delete\_key);  break;  case 2:  cout << "Введите номер рейса. ";  delete\_key = get\_user\_input\_with\_conditions(0, "natural\_number");  break;  case 6:  cout << "Введите число проданных билетов. ";  delete\_key = get\_user\_input\_with\_conditions(0, "natural\_number");  break;  case 7:  cout << "Введите расчетное число пассажиров. ";  delete\_key = get\_user\_input\_with\_conditions(0, "natural\_number");  break;  default:  cout << "error";  return;  } // Конец блока "В зависимости от номера параметра удаления, считываем параметр удаления"  cout << "\nУдаляем автобусы по заданным критериям...\n\n";  bool flag = false; // Был ли удален хотя бы один автобус  bool deliting = false; // Удаляем ли этот автобус    // Линейный односвязный список, удаление выбранных элементов:  if (bs->bus\_station\_type\_of\_list == '1') {  // Создадим переменную-автобуса для "движения" по списку:  bus\_from\_linear\_singly\_linked\_list\* iterator\_bus = new bus\_from\_linear\_singly\_linked\_list;  // Установим ее до начала списка, т.е. она будет указывать на первый элемент списка.  // Таким образом, удалять мы будем не текущий элемент, а следующий, но так же переберем все автобусы из списка, ведь  // начали не с первого элемента, а с "добавленного" перед первым элемента.  iterator\_bus->bus\_next\_bus = bs->bus\_station\_buses\_linear\_singly\_linked\_list->buses\_first\_bus;  // Цикл прохода по списку:  while (iterator\_bus->bus\_next\_bus != NULL) {  // В зависимости от номера параметра удаления, делаем соответствующее сравнение с данными автобуса:  switch (delete\_parameter) {  case 1: // Номер параметра удаления соответствует "Направление"  // Если у следующего автобуса "Направление" равно удаляемому пользователям, то удаляем следующий автобус:  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_destination) deliting = true;  break;  case 2: // Аналогично ...  // Аналогично...  if (stoi(delete\_key) == iterator\_bus->bus\_next\_bus->bus\_data->bus\_passage\_number) deliting = true;  break;  case 3: // ...  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_brand) deliting = true;  break;  case 4:  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_departure\_time) deliting = true;  break;  case 5:  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_full\_name\_of\_the\_driver) deliting = true;  break;  case 6:  if (stoi(delete\_key) == iterator\_bus->bus\_next\_bus->bus\_data->bus\_number\_of\_tickets\_sold) deliting = true;  break;  case 7:  if (stoi(delete\_key) == iterator\_bus->bus\_next\_bus->bus\_data->bus\_estimated\_number\_of\_passengers) deliting = true;  break;  default:  cout << "Error in delete function";  exit(1);  } // Конец блока "В зависимости от номера параметра удаления, делаем соответствующее сравнение с данными автобуса"  // Удаление автобуса  if (deliting) {    cout << "\nУдаляем автобус: ";  // Печатаем автобус:  output\_bus\_to\_console(iterator\_bus->bus\_next\_bus);  // Если удаляемый автобус - это первый в списке:  if (iterator\_bus->bus\_next\_bus == bs->bus\_station\_buses\_linear\_singly\_linked\_list->buses\_first\_bus) {  // Сдвинем указатель на начало списка на следующий автобус (или на NULL):  bs->bus\_station\_buses\_linear\_singly\_linked\_list->buses\_first\_bus = iterator\_bus->bus\_next\_bus->bus\_next\_bus;  }  // Удаляем автобус:  delete\_next\_bus(iterator\_bus);  flag = true; // Был удален хоть один автобус  deliting = false; // Этот флаг должен стоять false в начале каждого прохода цикла, он опредяляет, будет ли удаление  } // Конец блока "Удаление автобуса"  // Идем дальше по списку  iterator\_bus = iterator\_bus->bus\_next\_bus;  } // Конец блок "Цикл прохода по списку"  } // Конец блока "Линейный односвязный список, удаление выбранных элементов"  // Линейный двухсвязный список, удаление выбранных элементов:  else {  // Создадим переменную-автобуса для "движения" по списку:  bus\_from\_linear\_doubly\_linked\_list\* iterator\_bus = new bus\_from\_linear\_doubly\_linked\_list;  // Установим ее до начала списка, т.е. она будет указывать на первый элемент списка.  // Таким образом, удалять мы будем не текущий элемент, а следующий, но так же переберем все автобусы из списка, ведь  // начали не с первого элемента, а с "добавленного" перед первым элемента.  iterator\_bus->bus\_next\_bus = bs->bus\_station\_buses\_linear\_doubly\_linked\_list->buses\_first\_bus;  // Цикл прохода по списку:  while (iterator\_bus->bus\_next\_bus != NULL) {  // В зависимости от номера параметра удаления, делаем соответствующее сравнение с данными автобуса:  switch (delete\_parameter) {  case 1: // Номер параметра удаления соответствует "Направление"  // Если у следующего автобуса "Направление" равно удаляемому пользователям, то удаляем следующий автобус:  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_destination) deliting = true;  break;  case 2: // Аналогично ...  // Аналогично...  if (stoi(delete\_key) == iterator\_bus->bus\_next\_bus->bus\_data->bus\_passage\_number) deliting = true;  break;  case 3: // ...  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_brand) deliting = true;  break;    case 4:  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_departure\_time) deliting = true;  break;    case 5:  if (delete\_key == iterator\_bus->bus\_next\_bus->bus\_data->bus\_full\_name\_of\_the\_driver) deliting = true;  break;  case 6:  if (stoi(delete\_key) == iterator\_bus->bus\_next\_bus->bus\_data->bus\_number\_of\_tickets\_sold) deliting = true;  break;  case 7:  if (stoi(delete\_key) == iterator\_bus->bus\_next\_bus->bus\_data->bus\_estimated\_number\_of\_passengers) deliting = true;  break;  default:  cout << "Error in delete function";  exit(1);  } // Конец блока "В зависимости от номера параметра удаления, делаем соответствующее сравнение с данными автобуса"  // Удаление автобуса:  if (deliting) {  cout << "\nУдаляем автобус: ";  // Печатаем автобус:  output\_bus\_to\_console(iterator\_bus->bus\_next\_bus);  // Если удаляемый автобус - это первый в списке:  if (iterator\_bus->bus\_next\_bus == bs->bus\_station\_buses\_linear\_doubly\_linked\_list->buses\_first\_bus) {  // Сдвинем указатель на начало списка на следующий автобус (или на NULL):  bs->bus\_station\_buses\_linear\_doubly\_linked\_list->buses\_first\_bus = iterator\_bus->bus\_next\_bus->bus\_next\_bus;  }  // Удаляем автобус:  delete\_next\_bus(iterator\_bus);  flag = true;  deliting = false;  } // Конец блока "Удаление автобуса"  // Идем дальше по списку  iterator\_bus = iterator\_bus->bus\_next\_bus;  } // Конец блок "Цикл прохода по списку"  } // Конец блока "Линейный двухсвязный список, удаление выбранных элементов"  // Если не был удален ни один автобус:  if (!flag) cout << "\nНи один автобусов не был удален...\n\n";  // Если был удален хотя бы один автобус, придется обновить содержимое файла хранения базы данных:  else {  // Линейный односвязный список:  if (bs->bus\_station\_type\_of\_list == '1') save\_all\_buses\_to\_storage\_file(bs->bus\_station\_buses\_linear\_singly\_linked\_list);  // Линейный двухсвязный список:  else save\_all\_buses\_to\_storage\_file(bs->bus\_station\_buses\_linear\_doubly\_linked\_list);  }  } // Конец функции "Удаление выбранных данных" |
| --- |

#### Файл group\_delete.cpp:

| // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // --------------------------------------------------------------------------------------------  // ----------------------------------- Курсовая работа ----------------------------------------  // --------------------------------------------------------------------------------------------  // -------Файл определения функций, содержащих главную часть алгоритма удаления данных---------  // --------------------------------------------------------------------------------------------  // Подключение стандартных библиотек:  #include <fstream>  // Подключение стандартного пространста имен:  using namespace std;  // Подключение других файлов программы:  #include "structures\_definition.h"  #include "constants.h"  // Объявление функций:  // Удаление всех данных (линейный односвязный список)  void delete\_all\_data(buses\_linear\_singly\_linked\_list\* buses);  // Удаление всех данных (линейный двухсвязный список)  void delete\_all\_data(buses\_linear\_doubly\_linked\_list\* buses);  // Удалить следующий автобус в линейном односвязном списке:  void delete\_next\_bus(bus\_from\_linear\_singly\_linked\_list\* bus);  // Удалить следующий автобус в линейном двухсвязном списке:  void delete\_next\_bus(bus\_from\_linear\_doubly\_linked\_list\* bus);  // Очистка файла хранения данных:  void clear\_storage\_file();  // Очистка файла выходных данных:  void clear\_output\_file();  // Определение функций:  // Удаление всех данных (линейный односвязный список):  void delete\_all\_data(buses\_linear\_singly\_linked\_list\* buses) {  // Очистка файла хранения данных (где хранится БД):  clear\_storage\_file();  // Очистка файла выходных данных:  clear\_output\_file();  // Создадим переменную-автобус для "движения" по списку:  bus\_from\_linear\_singly\_linked\_list\* iterator\_bus = buses->buses\_first\_bus; // Установлен на начало списка (может быть NULL)  // Если в списке нет ни одного автобуса, то работа функции завершена:  if (iterator\_bus == NULL) return;  // Удаляем запись о первом автобусе:  buses->buses\_first\_bus = NULL;  // Перебираем автобусы:  while (iterator\_bus->bus\_next\_bus != NULL) { // Пока еще есть автобусы перед данным  delete\_next\_bus(iterator\_bus); // Удаляем следующий автобус (то есть список "стягивается" к началу)  } // Конец блока "Перебираем автобусы"  // Удаляем содержимое ставшего последним автобуса:  iterator\_bus->bus\_data = NULL;  } // Конец блока "Удаление всех данных (линейный односвязный список)"  // Удаление всех данных (линейный двухсвязный список):  void delete\_all\_data(buses\_linear\_doubly\_linked\_list\* buses) {  // Очистка файла хранения данных (где хранится БД):  clear\_storage\_file();  // Очистка файла выходных данных:  clear\_output\_file();  // Создадим переменную-автобус для "движения" по списку:  bus\_from\_linear\_doubly\_linked\_list\* iterator\_bus = buses->buses\_first\_bus; // Установлен на начало списка (может быть NULL)  // Если в списке нет ни одного автобуса, то работа функции завершена:  if (iterator\_bus == NULL) return;  // Удаляем запись о первом и последнем автобусах:  buses->buses\_first\_bus = NULL;  buses->buses\_last\_bus = NULL;  // Перебираем автобусы:  while (iterator\_bus->bus\_next\_bus != NULL) { // Пока еще есть автобусы перед данным  delete\_next\_bus(iterator\_bus); // Удаляем следующий автобус (то есть список "стягивается" к началу)  } // Конец блока "Перебираем автобусы"  // Удаляем содержимое ставшего последним автобуса:  iterator\_bus->bus\_data = NULL;  // Открепляем последний элемент списка от списка и удаляем его содержимое:  iterator\_bus->bus\_previous\_bus = NULL;  iterator\_bus->bus\_data = NULL;  } // Конец блока "Удаление всех данных (линейный двухсвязный список)"  // Удалить следующий автобус в линейном односвязном списке:  void delete\_next\_bus(bus\_from\_linear\_singly\_linked\_list\* bus) {  // Объявление переменных:  bus\_from\_linear\_singly\_linked\_list\* tmp\_bus; // Дополнительный автобус  // Если данного автобуса нет, то делать нечего:  if (bus == NULL) return;    // Если следующего автобуса нет, то делать нечего:  if (bus->bus\_next\_bus == NULL) return;  // Очистим данные из следующего автобуса:  bus->bus\_next\_bus->bus\_data = NULL;  // Запомним следующий за следующим автобус:  tmp\_bus = bus->bus\_next\_bus->bus\_next\_bus;  // Очистим из следующего автобуса указатель на следующий за следующим автобус:  bus->bus\_next\_bus->bus\_next\_bus = NULL;  // Теперь данный автобус ведет в tmp\_bus:  bus->bus\_next\_bus = tmp\_bus;  } // Конец блока "Удалить следующий автобус в линейном односвязном списке"  // Удалить следующий автобус в линейном двухсвязном списке:  void delete\_next\_bus(bus\_from\_linear\_doubly\_linked\_list\* bus) {  // Объявление переменных:  bus\_from\_linear\_doubly\_linked\_list\* tmp\_bus; // Дополнительный автобус  // Если данного автобуса нет, то делать нечего:  if (bus == NULL) return;  // Если следующего автобуса нет, то делать нечего:  if (bus->bus\_next\_bus == NULL) return;  // Очистим данные из следующего автобуса:  bus->bus\_next\_bus->bus\_data = NULL;  // Запомним следующий за следующим автобус:  tmp\_bus = bus->bus\_next\_bus->bus\_next\_bus;  // Очистим из следующего автобуса указатели на соседние автобусы:  bus->bus\_next\_bus->bus\_next\_bus = NULL;  bus->bus\_next\_bus->bus\_previous\_bus = NULL;  // Теперь данный автобус ведет в tmp\_bus, а перед tmp\_bus стоит данный автобус:  bus->bus\_next\_bus = tmp\_bus;  tmp\_bus->bus\_previous\_bus = bus;  } // Конец блока "Удалить следующий автобус в линейном двухсвязном списке"  // Очистка файла хранения данных:  void clear\_storage\_file() {  // Объявление переменных:  ofstream storage\_file; // Поток вывода в файл  // Открываем файл хранения данных в режиме записи, тем самым стирая его содержимое:  storage\_file.open(gl\_bus\_station\_storage\_file, std::ofstream::out | std::ofstream::trunc);  // Закрываем файл:  storage\_file.close();  } // Конец блока "Очистка файла хранения данных"  // Очистка файла выходных данных:  void clear\_output\_file() {  // Объявление переменных:  ofstream output\_file; // Поток вывода в файл  // Открываем файл выходных данных в режиме записи, тем самым стирая его содержимое:  output\_file.open(gl\_output\_file, std::ofstream::out | std::ofstream::trunc);  // Закрываем файл:  output\_file.close();  } // Конец блока "Очистка файла выходных данных" |
| --- |

# Выводы

При выполнении задания был оттренирован навык построения большой по структуре программы так, чтобы код был организованным и масштабируемым.

Разработка происходила в последней версии программы Microsoft Visual Studio, применялись библиотеки стандартной библиотеки.

Была реализована БД в соответствии с заданием. Все опции из задания реализованы, программа работает для двух типов списка. Проведены тесты, подтверждающие работоспособность программы. Представлены блок-схемы, код (полная версия кода прикладывается); описание интерфейса позволяет разобраться в ее работе. Задание можно считать успешно выполненным.

\