

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет непрерывного и дистанционного обучения

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (Часть 2)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту  
на тему:

«Программное средство каталога автомобилей»

БГУИР КП 1-40 01 01 5910036 ПЗ

Студент: гр. 491052 Дубинин А. В.

Руководитель: асс. Шостак Е. В.

Минск 2017

## ОГЛАВЛЕНИЕ

1. Постановка задачи .....	3
2. Теоретические сведения .....	4
3. Модульная структура программы и ее описание .....	5
3. 1. Модуль main .....	5
3. 2. Пакет application .....	5
3. 3. Модуль handler .....	7
3. 4. Модуль view .....	8
4. Схемы алгоритмов решения задачи и их описание .....	9
4. 1. Алгоритм функции модуля application .....	9
4. 2. Алгоритмы функций модуля handler .....	10
4. 3. Алгоритмы функций модуля view .....	21
5. Руководство пользователя .....	26
5. 1. Системные требования .....	26
5. 2. Инструкция пользователя .....	26
6. Список литературы .....	35
7. Листинг программы .....	37
7. 1. Листинг исходного кода модуля main .....	37
7. 2. Листинг исходных кодов модулей пакета application .....	37
7. 3. Листинг исходного кода модуля handler .....	42
7. 4. Листинг исходного кода модуля view .....	50

## 1. ПОСТАНОВКА ЗАДАЧИ

Используя динамические списки, написать программу для хранения информации о поступивших в продажу автомобилях. Каждый элемент списка должен иметь следующие поля: марку автомобиля и его параметры: стоимость, расход бензина на 100 км, надежность (число лет безотказной работы), комфортность в баллах.

Покупатель в свою очередь имеет ряд требований по каждому из этих параметров. Эти требования задаются в виде интервала (например, стоимость — от 10 до 30 тысяч долларов, комфортность — от 8 до 10 баллов и т.п.). Требования вводить с клавиатуры.

В программе должны присутствовать следующие процедуры:

1. Формирование динамического списка.
2. Вывод списка на экран.
3. Добавление элемента в список.
4. Удаление элемента из списка.
5. Поиск в соответствии с требованиями покупателя.
6. Поиск элемента списка по соответствующему полю.
7. Сортировка по алфавиту.
8. Запись в файл и загрузка из файла.

Для записи и загрузки из файла использовать типизированный файл. Взаимодействие с пользователем осуществляется через консоль в ОС Linux.

## **2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

В настоящее время для автоматизации сбора и обработки информации на предприятиях используются различные программные комплексы и средства. Так, например, для деятельности салонов по продаже автомобилей и автодилеров внедряются программные решения, позволяющие сэкономить время работников и руководителей, которое тратится на привычную деятельность, и, как следствие, повысить производительность труда.

Данные программные средства позволяют:

- систематизировать автомобили в единый удобный каталог;
- вести систему учета операций поступления и продажи автомобилей;
- создавать карточки автомобилей;
- осуществлять быстрый поиск по наличию, стоимости, наименованию и характеристикам автотранспортных средств.

Так как основной задачей автосалонов и автодилеров является удовлетворение всех потребностей и запросов физических и юридических лиц на покупку автотранспортных средств, то благодаря отраслевым программным решениям для организаций подобного рода повышается уровень обслуживания клиентов.

### 3. МОДУЛЬНАЯ СТРУКТУРА ПРОГРАММЫ И ЕЕ ОПИСАНИЕ

Заданная функциональность программного средства каталога автомобилей реализована на языке программирования C. Компиляция и сборка программы выполнена с помощью GNU GCC Compiler в операционной системе Ubuntu 14.04 LTS 32-bit. Для реализации некоторых операций подключены следующие заголовочные файлы стандартной библиотеки языка C:

- *stdio.h* — для операций стандартного ввода и вывода, а также файловых операций;
- *stdlib.h* — для операций выделения и освобождения памяти (динамическое распределение памяти), а также контроля процесса выполнения программы;
- *string.h* — для операций с нуль-терминированными строками и различными функциями работы с памятью и строками.

Исходный код программы разбит на несколько модулей. Принцип модульности является средством упрощения задачи проектирования программного средства и распределения процесса разработки между группами разработчиков. При разбиении программы на модули для каждого из них указывается реализуемая им функциональность, а также связи с другими модулями. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы.

#### 3. 1. Модуль *main*

Модуль *main* содержит файл реализации *main.c* основной функции программы *main()*, служащей для запуска программы.

#### 3. 2. Пакет *application*

Пакет *application* содержит:

##### 3. 2. 1. Заголовочный файл *definitions.h*

В нем определены объявления макросов и констант для упрощения процесса сопровождения программы. Здесь следует выделить константу

*CATALOG\_FILE*, в которой объявлено наименование бинарного файла для хранения данных каталога автомобилей.

### 3. 2. 2. Заголовочный файл *types.h*

Объявлены необходимые структуры и новые типы.

Структура *vehicle* связывает характеристики автомобиля. Структура *node* описывает элемент списка автомобилей *list*. Тип *listPtr* — это указатель на список автомобилей. Тип *actionType* — указатель на функцию, принимающей в качестве параметра указатель на список автомобилей.

В качестве динамической структуры данных для работы со списком автомобилей выбран линейный однонаправленный связный список:

```
typedef struct node {  
    vehicle car;  
    struct node *next;  
} list;
```

Данный список представляет собой (рис. 3.2.2.1) последовательность связанных между собой узлов (*node*), каждый из которых состоит из поля данных (*car* типа *vehicle*) и указателя на место в памяти, где расположен следующий элемент списка (*\*next*). Последний узел списка указывает на NULL. Узел, на который нет указателя, является первым (головным) узлом списка.

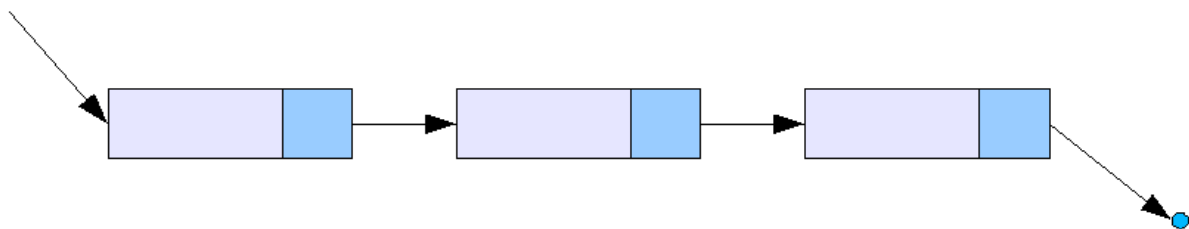


Рис. 3.2.2.1. Линейный однонаправленный связный список

Используя линейный однонаправленный связный список, можно не ограничиваться в количестве рабочих элементов, как, например, в массивах, а увеличивать или уменьшать их в процессе выполнения программы. Это дает возможность заранее не учитывать объем требуемой памяти и эффективно ее использовать.

### 3. 2. 3. Заголовочный файл *runner.h*

Содержит прототип функции *runApplication()*.

### 3. 2. 4. Файл реализации *runner.c*

В нем реализована функция *runApplication()*, которая выполняет следующие операции:

- открытие бинарного файла-хранилища данных каталога автомобилей;
- формирование линейного однонаправленного связного списка на основе данных из бинарного файла;
- вызов функции вывода на экран главного меню программы;
- вызов операций заданной функциональности (вывод на экран списка, добавление/удаление элементов в/из список, поиск элементов, сортировка элементов, запись элементов в файл).

## 3. 3. Модуль *handler*

Реализует функции основных операций заданной функциональности программного средства каталога автомобилей.

### 3. 3. 1. Заголовочный файл *handler.h*

Содержит прототипы функций.

### 3. 3. 2. Файл реализации *handler.c*

В нем реализованы функции:

- *doRepeatedAction()* — функция вызова повторяющихся операций программы;
- *formList()* — формирование линейного однонаправленного связного списка;
- *countListSize()* — подсчет количества элементов списка;
- *displayCarsList()* — вывод на экран списка;
- *addCarsIntoList()* — добавление нового элемента в список;

- *removeCarsFromList()* — удаление элемента из списка;
- *searchByWishes()* — поиска автомобилей в соответствии с требованиями покупателя;
- *searchByField()* — поиска автомобилей по соответствующему полю;
- *sortByAlphabet()* — сортировки (простыми обменами) списка;
- *saveCatalog()* — запись элементов в файл.

### 3. 4. Модуль *view*

Реализует функции пользовательского интерфейса программного средства каталога автомобилей, т. е. операции по выводу информации на экран и вводу данных пользователем.

#### 3. 4. 1. Заголовочный файл *view.h*

Содержит прототипы функций.

#### 3. 4. 2. Файл реализации *view.c*

В нем реализованы функции:

- *hr()* — вывод на экран горизонтальной полосы;
- *toUpCase()* — преобразование строки к верхнему регистру;
- *printString()* — вывод на экран строки;
- *renderTitle()*, *render####Title()*, *render####Msg()* — вывод на экран заголовков;
- *renderCarsListHeader()* — вывод на экран головной строки таблицы списка автомобилей;
- *renderCarsListRow()* — вывод на экран строки таблицы списка автомобилей;
- *renderSearchResult()* — вывода на экран строки таблицы списка найденных автомобилей;
- *renderSearchResultFooter()* — вывод на экран списка действий после поиска автомобилей;
- *returnToMainMenu()* — возврат в главное меню программы;
- *displayMainMenu()* — вывод на экран главного меню программы.



## 4. СХЕМЫ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ И ИХ ОПИСАНИЕ

### 4. 1. Алгоритм функции модуля *application*

Схема алгоритма функции *runApplication()* представлена на рис. 4.1.1. Данная функция вызывается из функции *main()* и служит для исполнения основных операций программы. Сначала происходит открытие бинарного файла, хранящего данные каталога автомобилей, с помощью функции *fopen()*, которая возвращает указатель *\*catalog* типа *FILE*. Далее формируется линейный однонаправленный связный список (указатель *index* типа *listPtr* — головной элемент списка) с помощью функции *formList(catalog)* и вызовом *fclose(catalog)* файл закрывается. Подсчитывается количество элементов из файла каталога (*catalogSize*) вызовом функции *countListSize(&index)*.

Для работы с главным меню программы используется цикл с постусловием *do ... while*, выход из которого происходит по флагу *isExit*. В каждой итерации цикла вычисляется кол-во элементов списка *listSize*, вызывается функция *displayMainMenu()* для вывода на экран главного меню программы. Выбранный пункт меню сохраняется в переменной *actionKey*, которая затем анализируется оператором *switch*. После завершения цикла освобождается память *free(index)*.

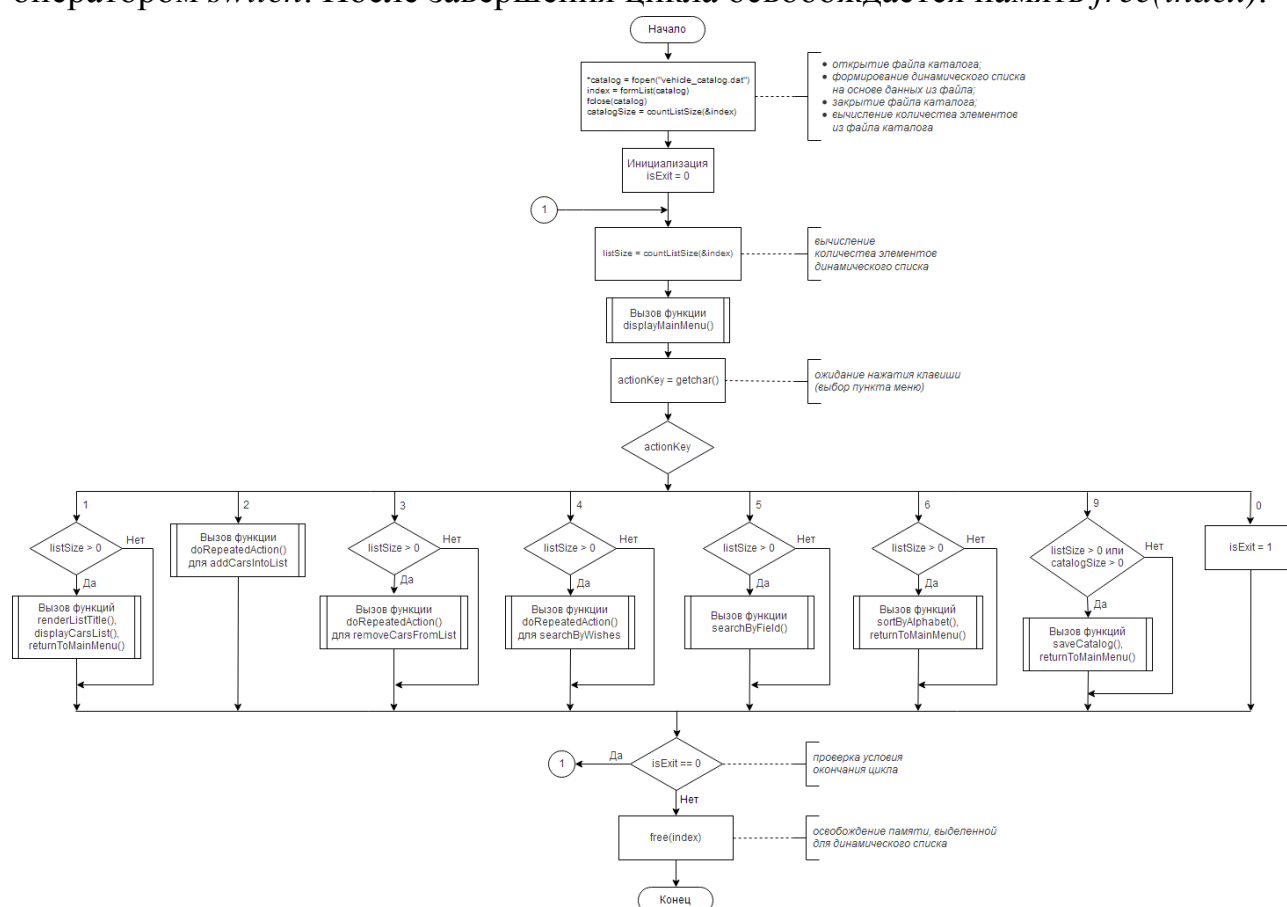


Рис. 4.1.1. Схема алгоритма функции *runApplication()*

## 4. 2. Алгоритмы функций модуля *handler*

Функция вызова повторяющихся операций программы *doRepeatedAction()* (рис. 4.2.1) принимает аргументами указатель на заглавный элемент списка (*\*index* типа *listPtr*) и указатель на функцию (*action* типа *actionType*). Сначала происходит вызов функции *action(index)*. Затем в цикле с постусловием *do ... while* в каждой итерации анализируется значение нажатой клавиши *actionKey*: если оно равно «9», то происходит вызов функции *action(index)* и переход к следующей итерации; если оно равно «0», то происходит выход из цикла. Затем выполнение функции *doRepeatedAction()* завершается.

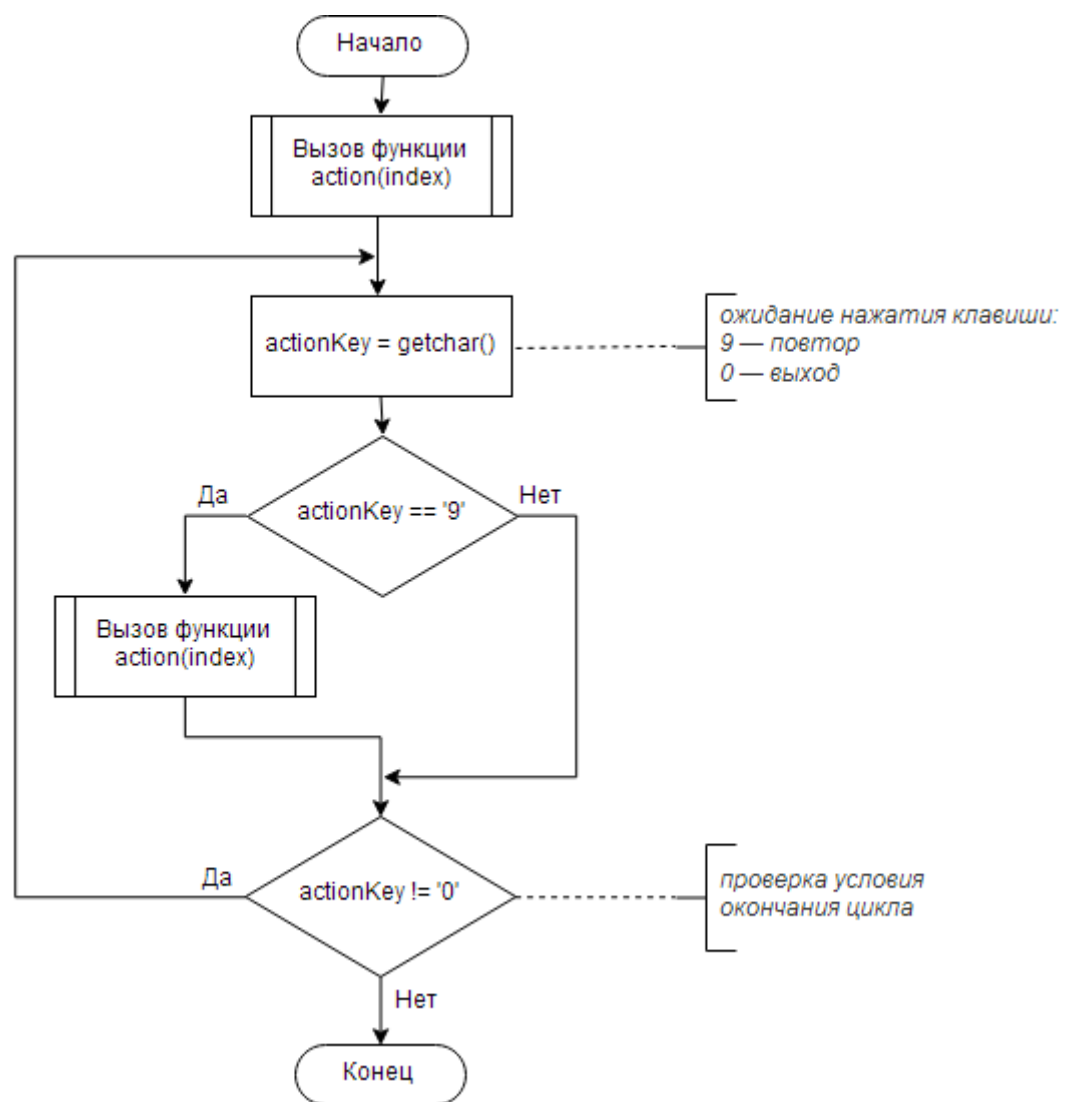


Рис. 4.2.1. Схема алгоритма функции *doRepeatedAction()*

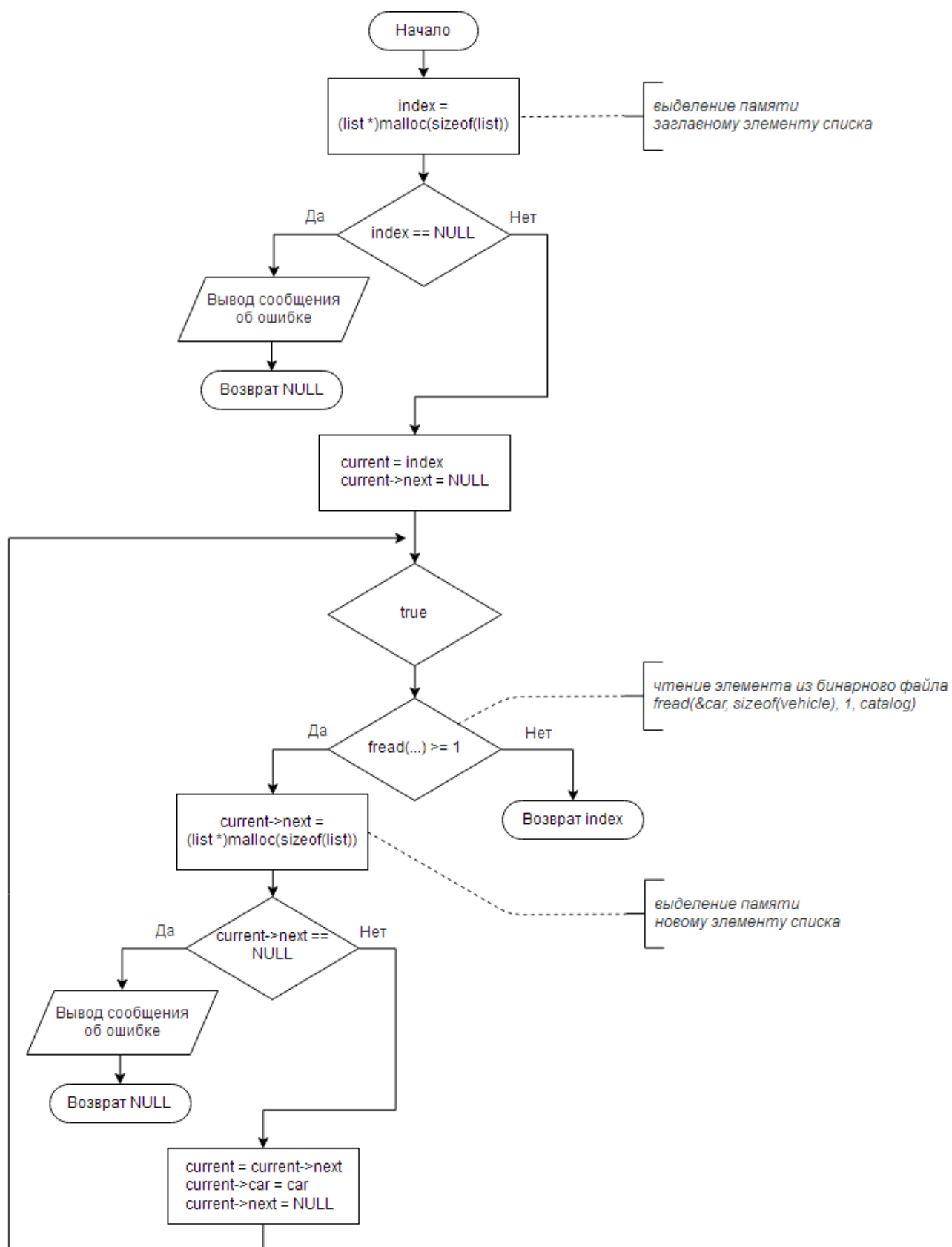


Рис. 4.2.2. Схема алгоритма функции *formList()*

Функция формирования линейного однонаправленного связанного списка *formList()* (рис. 4.2.2) принимает аргументом указатель *\*catalog* типа *FILE* и возвращает указатель типа *listPtr* на заглавный элемент списка, в случае его успешного формирования, либо *NULL* в случае ошибки. Сначала происходит выделение памяти заглавному элементу списка *index* с помощью функции *malloc()*, при этом, если произошла ошибка выделения памяти (*index == NULL*), то выводится на экран соответствующее сообщения посредством *puts()* и возврат *NULL*. Далее инициализируется рабочий элемент списка *current = index*, у которого указатель на следующий элемент равен *NULL*. Затем в каждой итерации цикла с предусловием *while* читается из бинарного файла в оперативную память структура *car* типа *vehicle* с помощью функции *fread()*, происходит выделением памяти следующему элементу списка *current->next*, данные элемента списка (*current->car*) инициализируются значением структуры *car*. Выход из цикла происходит после того, как прочитан из бинарного файла последний элемент.

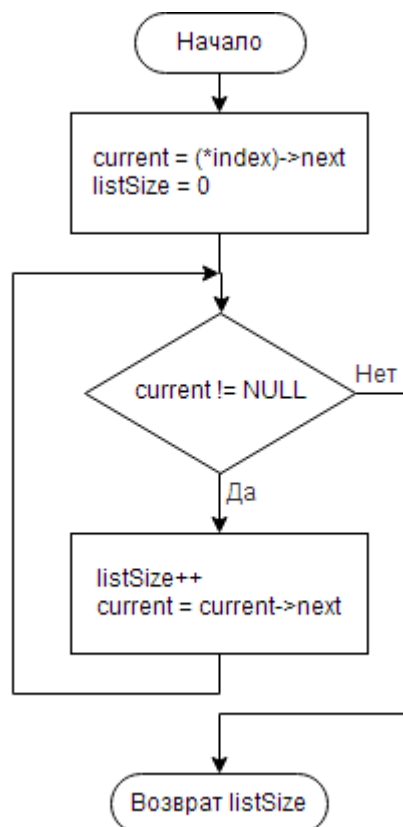


Рис. 4.2.3. Схема алгоритма функции *countListSize()*

Функция подсчета количества элементов списка *countListSize()* (рис. 4.2.3) принимает аргументом указатель на заглавный элемент списка (*\*index* типа *listPtr*) и возвращает количество его элементов. Подсчет происходит в цикле с

предусловием *while* до тех пор, пока указатель на следующий элемент списка не равен *NULL*. В каждой итерации цикла происходит инкремент счетчика *listSize* и перемещение на следующий элемент.

Функция вывода на экран списка автомобилей *displayCarsList()* (рис. 4.2.4) принимает аргументом указатель на заглавный элемент списка (*\*index* типа *listPtr*). Сначала происходит вызов функции *renderCarsListHeader()* для вывода

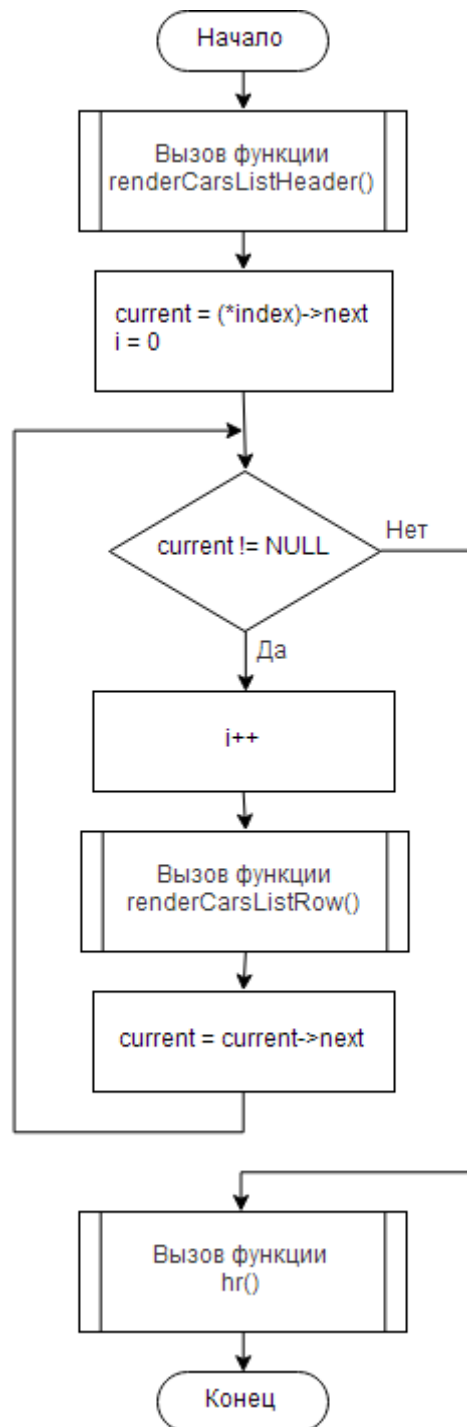


Рис. 4.2.4. Схема алгоритма функции *displayCarsList()*

на экран головной строки списка автомобилей. Затем вывод на экран данных каждого элемента списка происходит в цикле с предусловием *while* до тех пор, пока указатель на следующий элемент списка не равен *NULL*. В каждой итерации цикла вызывается функция *renderCarsListRow()*.

Функция добавление нового элемента в список автомобилей *addCarsIntoList()* (рис. 4.2.5) принимает аргументом указатель на заглавный элемент списка (*\*index* типа *listPtr*). Сначала вызывается функция *renderAddCarTitle()* для вывода заголовка диалогового окна. Затем пользователь вводит данные нового автомобиля *newCar* типа *vehicle* с помощью функции *scanf()*. С помощью цикла с предусловием *while* происходит перемещение в конец списка, где у последнего элемента указателю на следующий элемент списка присваивается вновь созданный (с помощью *malloc()*) элемент *newNode*. Данные элемента *newNode->car* инициализируются значением структуры нового автомобиля *newCar*. Далее выводится сообщение об успешном добавлении нового автомобиля в список с помощью *printf()*, а также предложение о продолжении добавления очередного нового элемента либо возврат в главное меню программы.

Функция удаления элемента из списка автомобилей *removeCarsFromList()* (рис. 4.2.6) принимает аргументом указатель на заглавный элемент списка (*\*index* типа *listPtr*). Сначала вычисляется количество элементов списка *listSize* с помощью вызова функции *countListSize(index)*. Если *listSize == 0*, то выводится сообщение вызовом функции *renderEmptyListTitle()* о том, что список пуст. Если в списке имеются элементы (*listSize > 0*), то осуществляется вывод на экран данных всех элементов вызовом функции *displayCarsList(index)*. Далее пользователь вводит с помощью *scanf()* номер элемента списка *n*. После чего с помощью цикла *for* с параметром *i* происходит перемещение в списке до указанного элемента. При этом указателю на следующий элемент у предыдущего элемента присваивает указатель на следующий элемент удаляемого. Затем указанный элемент удаляется из динамической памяти с помощью функции *free()*. Далее осуществляется вывод на экран данных всех элементов вызовом функции *displayCarsList(index)*, где уже отсутствует удаленный элемент. Выводится сообщение об успешном удалении автомобиля из списка с помощью *printf()*, а также предложение о продолжении удаления элементов либо возврат в главное меню программы.

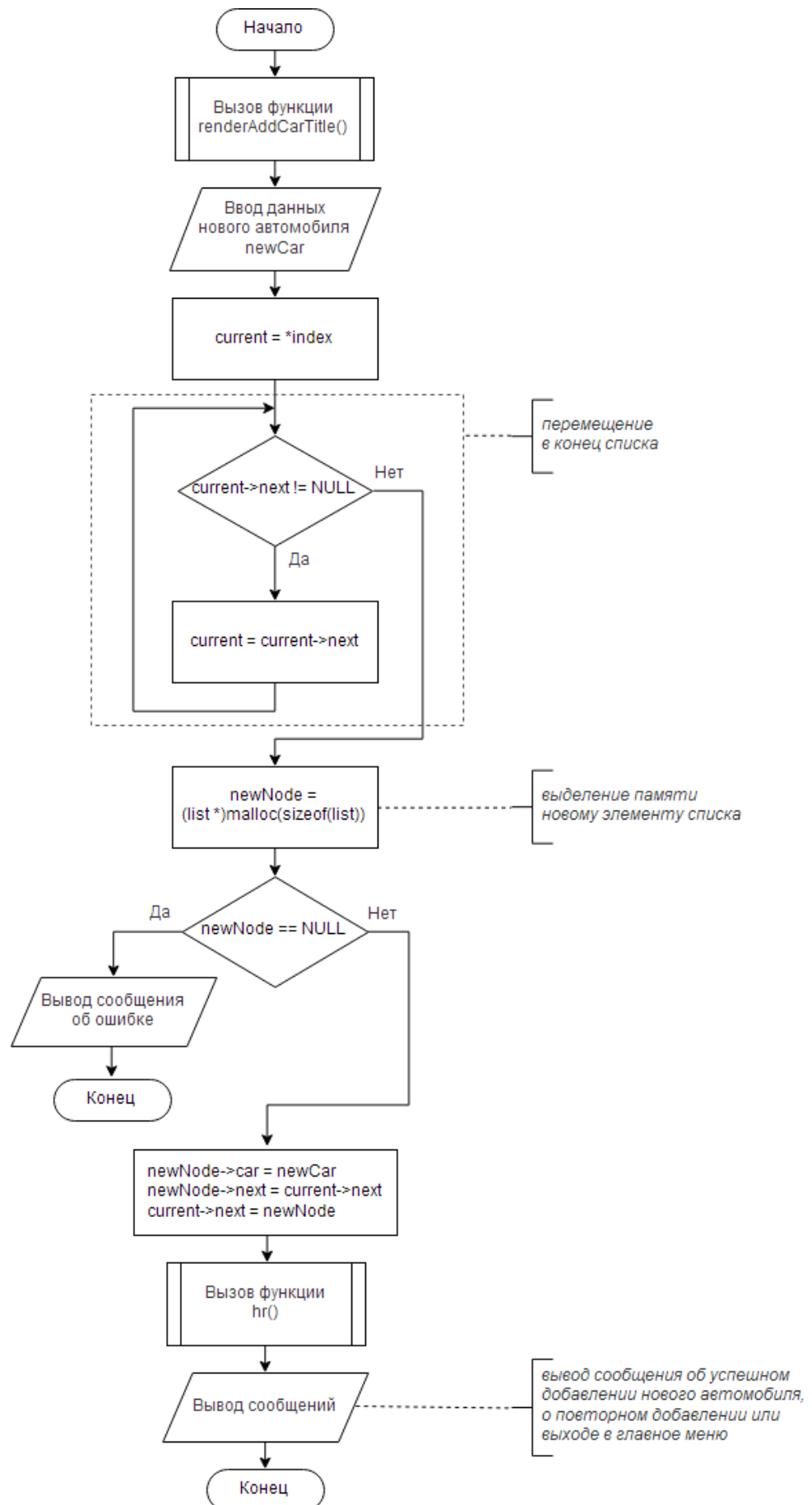


Рис. 4.2.5. Схема алгоритма функции *addCarsIntoList()*

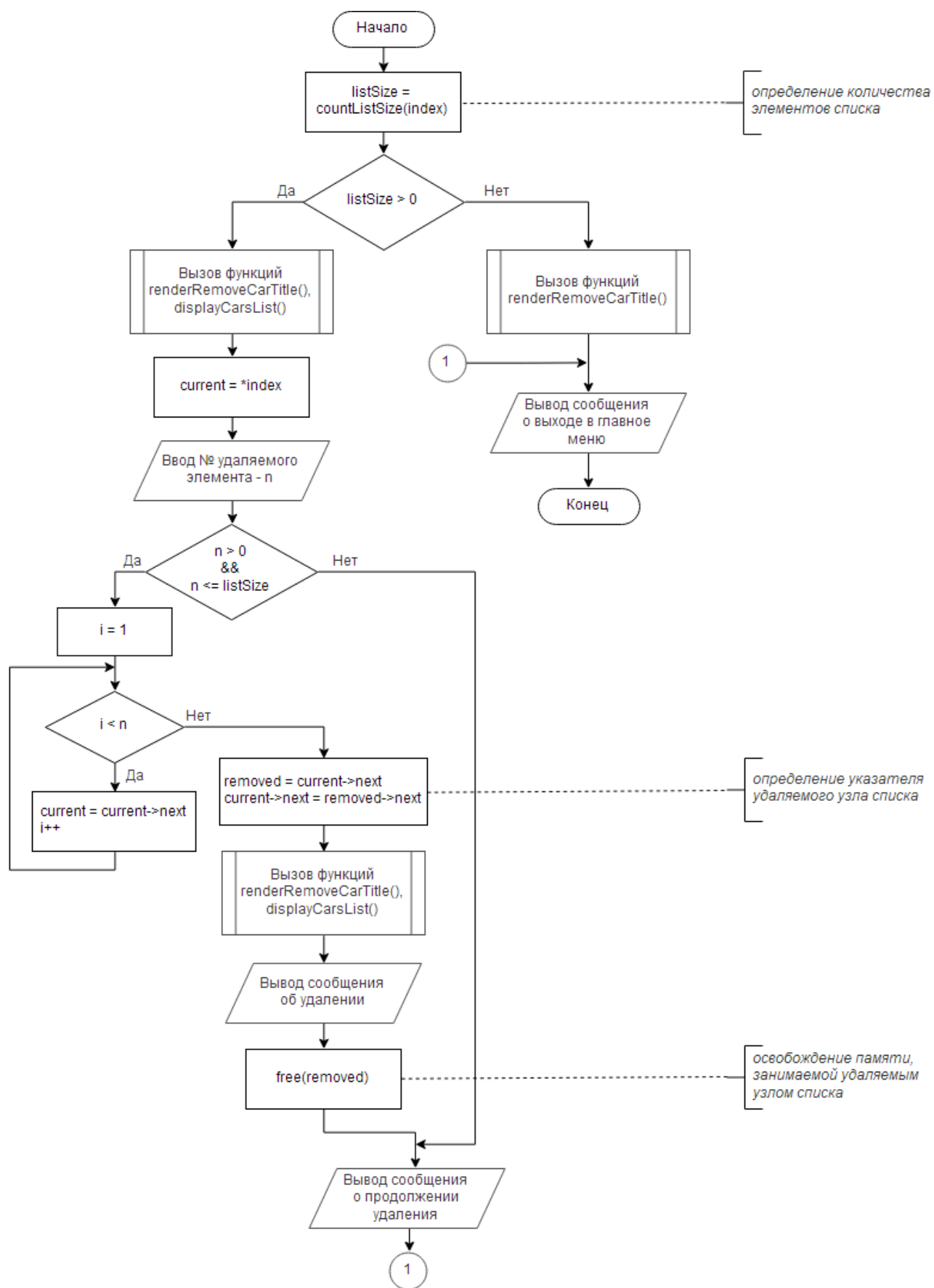


Рис. 4.2.6. Схема алгоритма функции *removeCarsFromList()*



Функция поиска автомобилей в соответствии с требованиями покупателя *searchByWishes()* (рис. 4.2.7) принимает аргументом указатель на заглавный элемент списка (*\*index* типа *listPtr*). Сначала вызывается функция *renderSearchByWishesTitle()* для вывода заголовка диалогового окна. Затем происходит ввод данных для поиска (минимальные и максимальные значения цены, расхода топлива, надежности и комфортности) с помощью функции *scanf()*. С помощью цикла с предусловием *while* происходит перемещение по списку, где в каждой итерации проверяется вхождение характеристик автомобиля из списка *car* в диапазон заданных пользователем значений. Если условие вхождения выполняется, то данные такого элемента списка выводятся на экран вызовом функции *renderSearchResult()*. Затем выводится предложение о продолжении поиска элементов либо возврат в главное меню программы.

Функция сортировки по алфавиту списка по марке автомобиля *sortByAlphabet()* (рис. 4.2.8) принимает аргументом указатель на заглавный элемент списка (*\*index* типа *listPtr*). Сортировка осуществляется простыми обменами (сортировка пузырьком). Обмены реализованы в цикле с постусловием *do ... while*, выходом из которого является признак окончания сортировки *isSorted*. В каждой итерации цикла сначала инициализируется флаг завершения сортировки *isSorted = 1*. Затем с помощью цикла с предусловием *while* происходит перемещение по списку, где в каждой итерации данного внутреннего цикла сравниваются значения марок автомобилей текущего элемента списка и следующего. Сравнение значений происходит без учета регистра с помощью функций *strcpy()* и *toUpCase()*. Если значение марки автомобиля текущего элемента больше значения марки автомобиля следующего элемента, то такие элементы в списке меняются местами, при этом флагу завершения сортировки *isSorted* присваивается значение *0*. После завершения сортировки выводится соответствующее сообщение вызовом функции *renderSortedMsg()*.

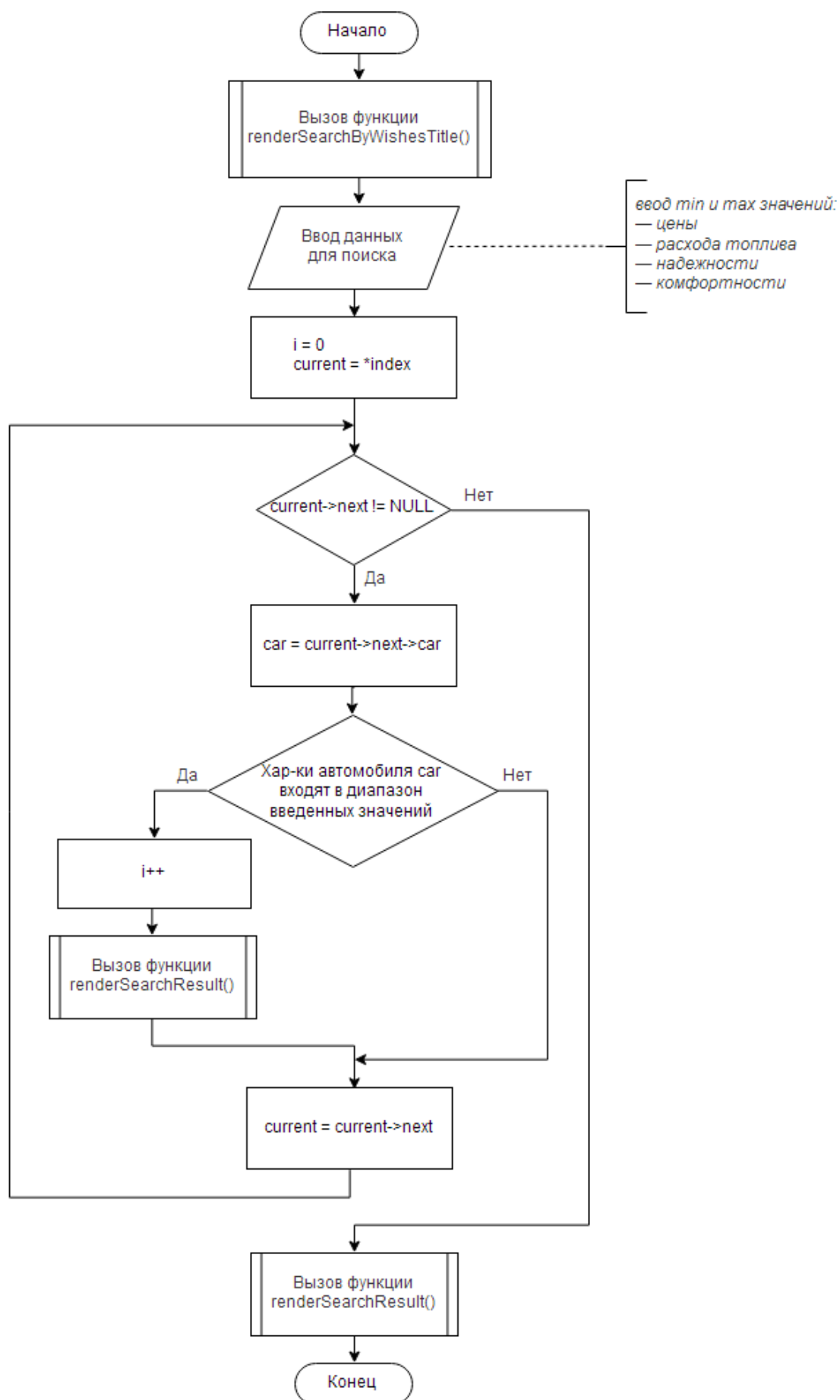


Рис. 4.2.7. Схема алгоритма функции *searchByWishes()*

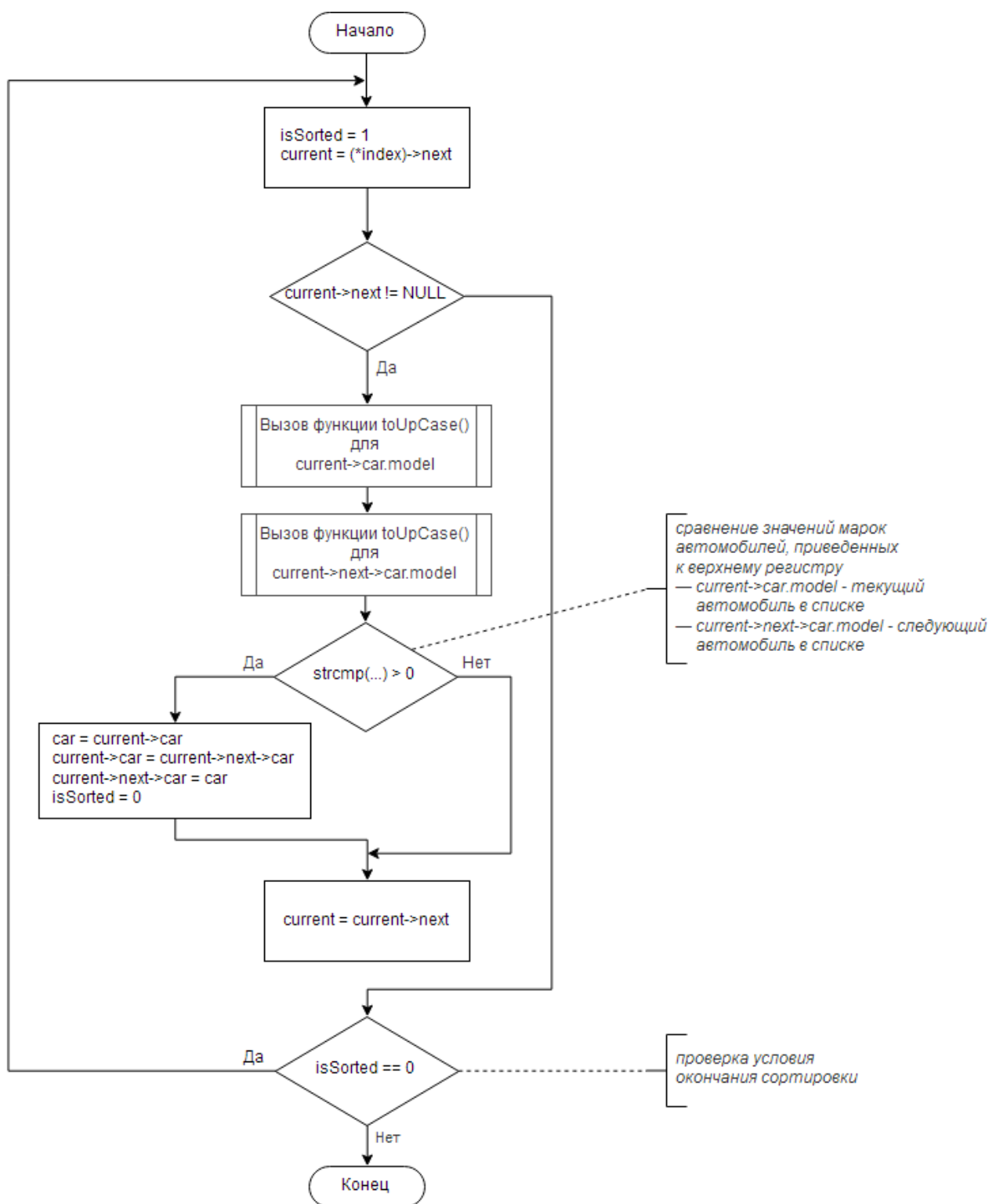


Рис. 4.2.8. Схема алгоритма функции *sortByAlphabet()*

Функция сохранения информации об автомобилях из динамического списка в бинарный файл каталога *saveCatalog()* (рис. 4.2.9) принимает аргументом указатель на заглавный элемент списка (*\*index* типа *listPtr*) и возвращает количество записанных в файл элементов списка. Сначала

происходит открытие бинарного файл с помощью функции *fopen()*, которая возвращает указатель *\*catalog* типа *FILE*. Затем инициализируется нулем значение переменной *catalogSize* — количество записанных в файл элементов. С помощью цикла с предусловием *while* происходит перемещение по списку, где в каждой итерации структура *car* типа *vehicle* записывается из памяти в бинарный файл вызовом функции *fwrite()*, а также инкрементируется значение переменной *catalogSize*. По завершению цикла вызовом *fclose(catalog)* файл закрывается. Выводится сообщение об успешном сохранении вызовом функции *renderSavedMsg()*.

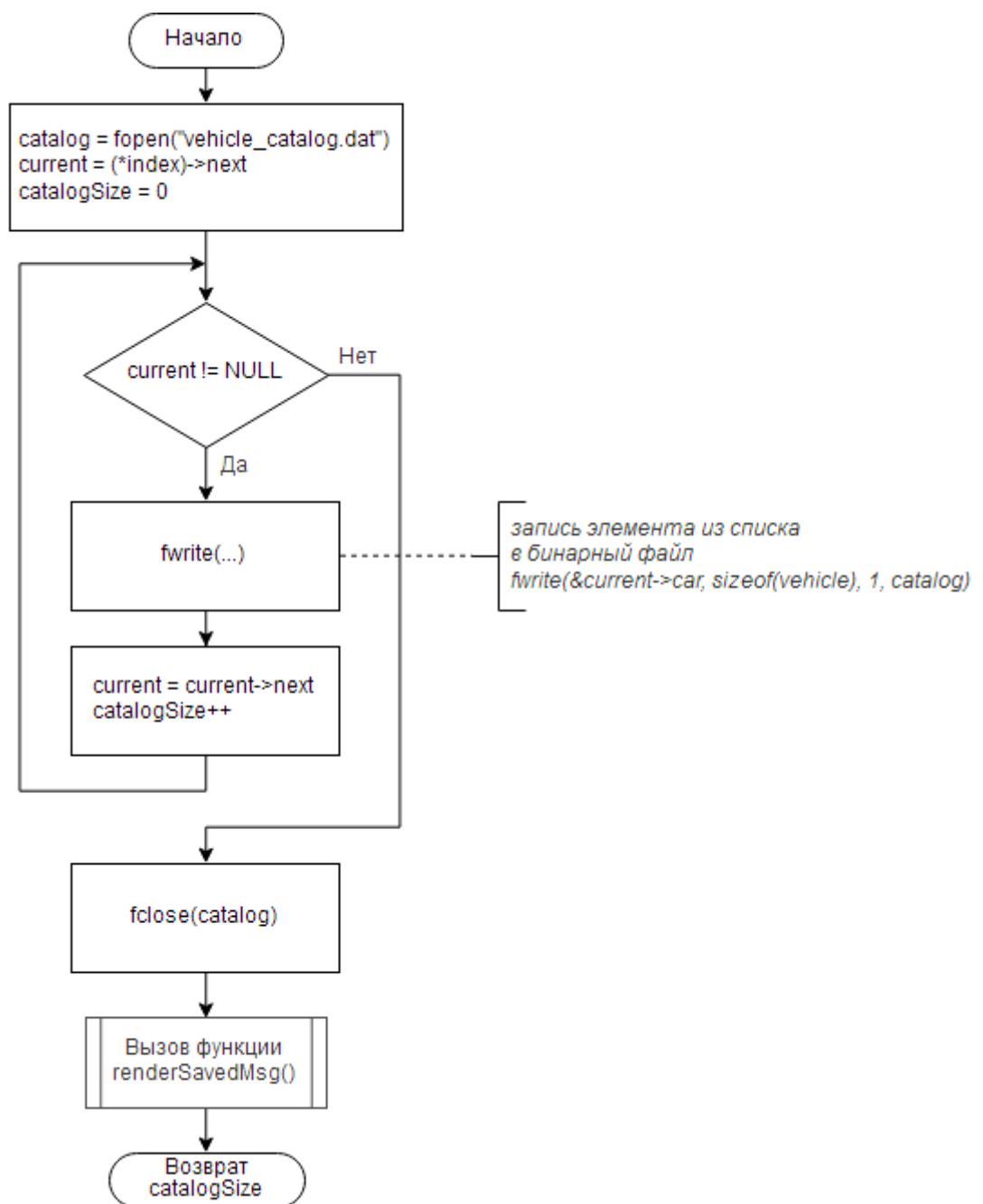


Рис. 4.2.9. Схема алгоритма функции *saveCatalog()*

### 4. 3. Алгоритмы функций модуля *view*

В функции вывода на экран горизонтальной полосы *hr()* (рис. 4.3.1) с помощью цикла *for* с параметром *i* в каждой его итерации выводится на экран символ «-» с помощью вызова функции *putchar()*. Количество итераций цикла ограничено константой *WINDOW\_WIDTH*.

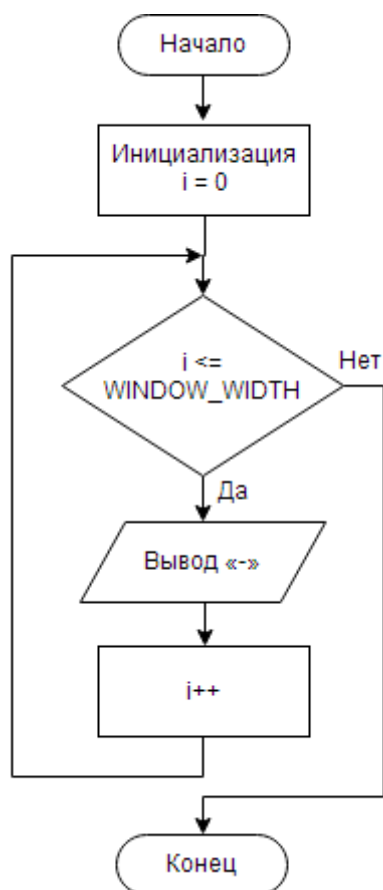


Рис. 4.3.1. Схема алгоритма функции *hr()*

Функция преобразования строки к верхнему регистру *toUpCase()* (рис. 4.3.2) принимает аргументом строку (*\*str* типа *char*) и с помощью цикла с параметром *for* в каждой его итерации преобразует к верхнему регистру каждый символ строки вызовом функции *toupper()*. Цикл заканчивается после обработки последнего символа строки.

Функция вывода на экран заголовка *renderTitle()* (рис. 4.3.3) принимает аргументом строку (*\*title* типа *char*). Сначала происходит очистка диалогового окна системным вызовом «*clear*» с помощью функции *system()*. Затем выводится на экран заголовок *title* с помощью функции *printf()*.

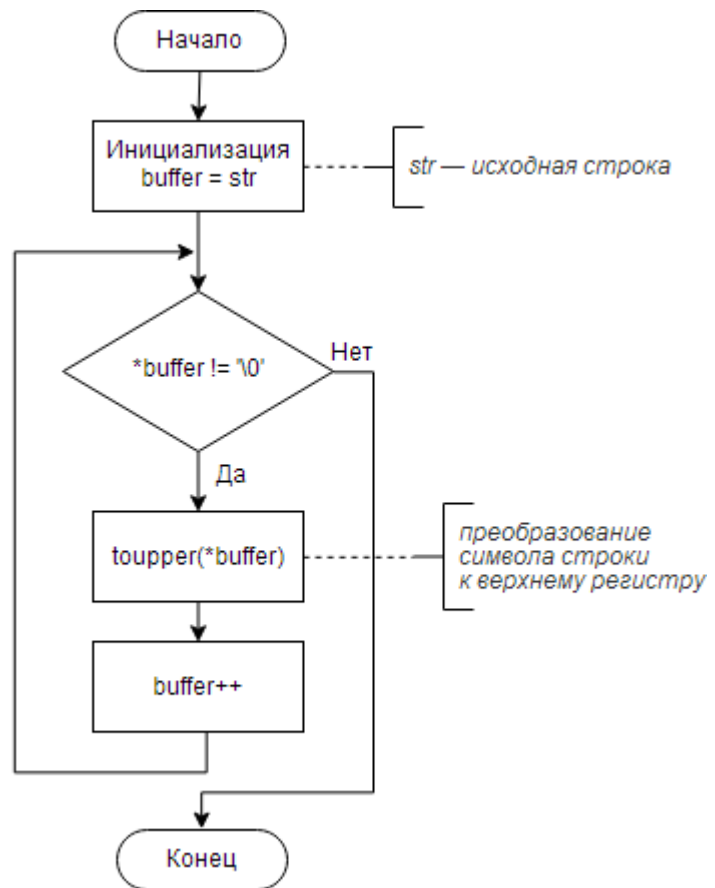


Рис. 4.3.2. Схема алгоритма функции *toUpCase()*

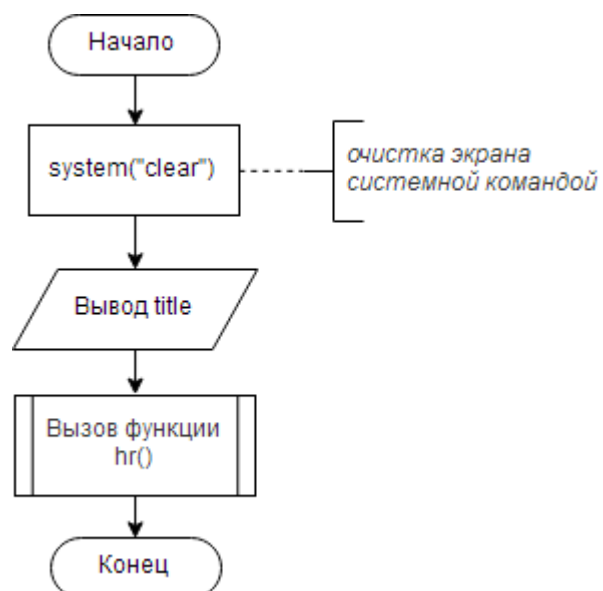


Рис. 4.3.3. Схема алгоритма функции *renderTitle()*

Функция вывода на экран одной строки списка найденных автомобилей *renderSearchResult()* (рис. 4.3.4) принимает аргументом номер строки (*number* типа *unsigned int*) и структуру *car* типа *vehicle*. Если это первый номер строки (*number == 1*), то на экран выводится заголовок диалогового окна и головная строка списка найденных автомобилей вызовами соответственно функций *renderSearchResultTitle()* и *renderCarsListHeader()*. Вызывается функция *renderCarsListRow()* для вывода на экран строки списка автомобилей.

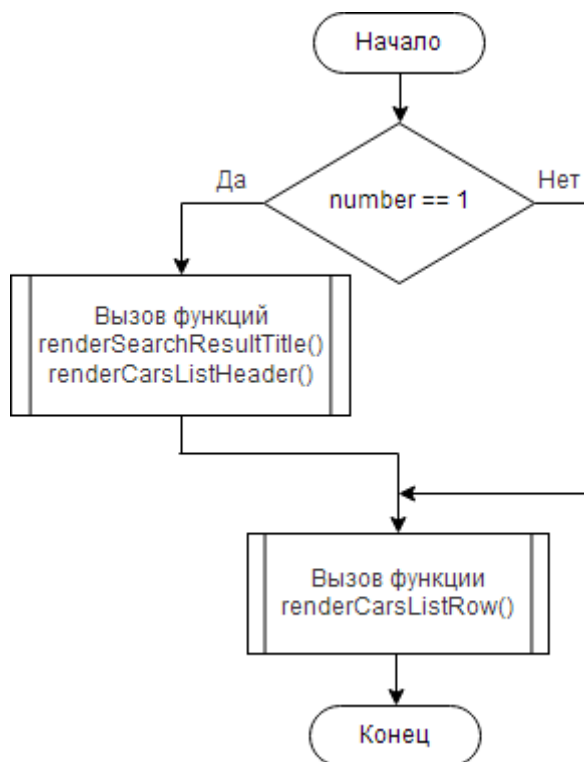


Рис. 4.3.4. Схема алгоритма функции *renderSearchResult()*

Функция вывода на экран выбора действия после поиска автомобилей *renderSearchResultFooter()* (рис. 4.3.5) принимает аргументом количество найденных автомобилей (*number* типа *unsigned int*). Если *number == 0*, то выводится сообщение, что автомобили не найдены, вызовом функции *renderNotFoundMsg()*. Далее с помощью функций *puts()* выводятся сообщения о повторном поиске или о выходе в главное меню.

В функции, реализующей возврат в главное меню программы *returnToMainMenu()* (рис. 4.3.6), сначала выводится сообщение о выходе в главное меню, затем с помощью цикла с предусловием *while* и вызова функции *getchar()* в каждой его итерации ожидается нажатие клавиши «0», которое завершает цикл.

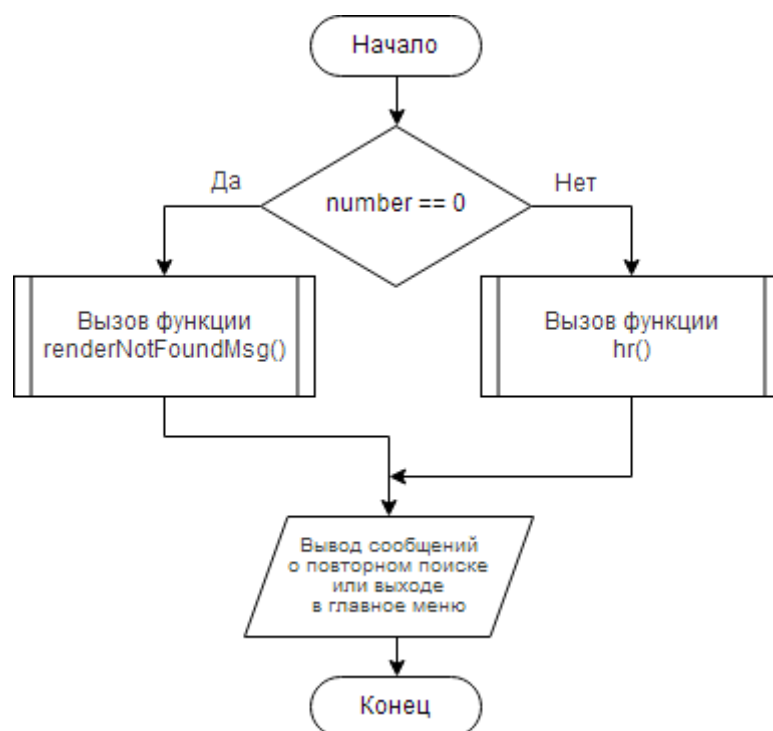


Рис. 4.3.5. Схема алгоритма функции *renderSearchResultFooter()*

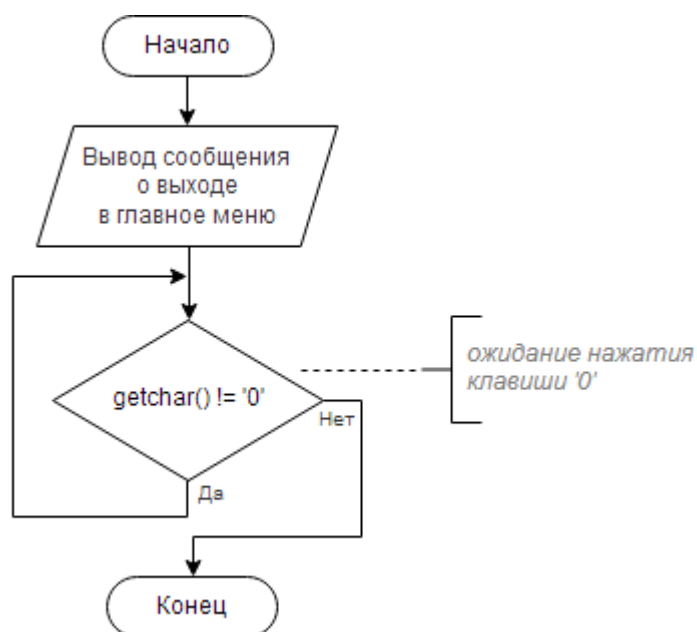


Рис. 4.3.6. Схема алгоритма функции *returnToMainMenu()*

Функция вывода на экран главного меню программы *displayMainMenu()* (рис. 4.3.7) принимает аргументом количество элементов динамического списка (*listSize* типа *unsigned int*) и количество элементов бинарного файла каталога (*catalogSize* типа *unsigned int*). Сначала происходит очистка диалогового окна системным вызовом «*clear*» с помощью функции *system()*. Если *listSize* > 0, то вызовом функции *renderListTitle()* выводится заголовок



диалогового окна и осуществляется вывод на экран всех пунктов меню программы с помощью функций *puts()*. Если *listSize == 0*, то заголовок диалогового окна выводится функцией *renderEmptyListTitle()* и на экран с помощью функций *puts()* выводится только один пункт меню о добавлении нового автомобиля в каталог. Вывод пункта меню о сохранение списка в файл возможен, если *listSize > 0* или *catalogSize > 0*. Далее вызовом функции *puts()* осуществляется вывод пункта меню выхода из программы.

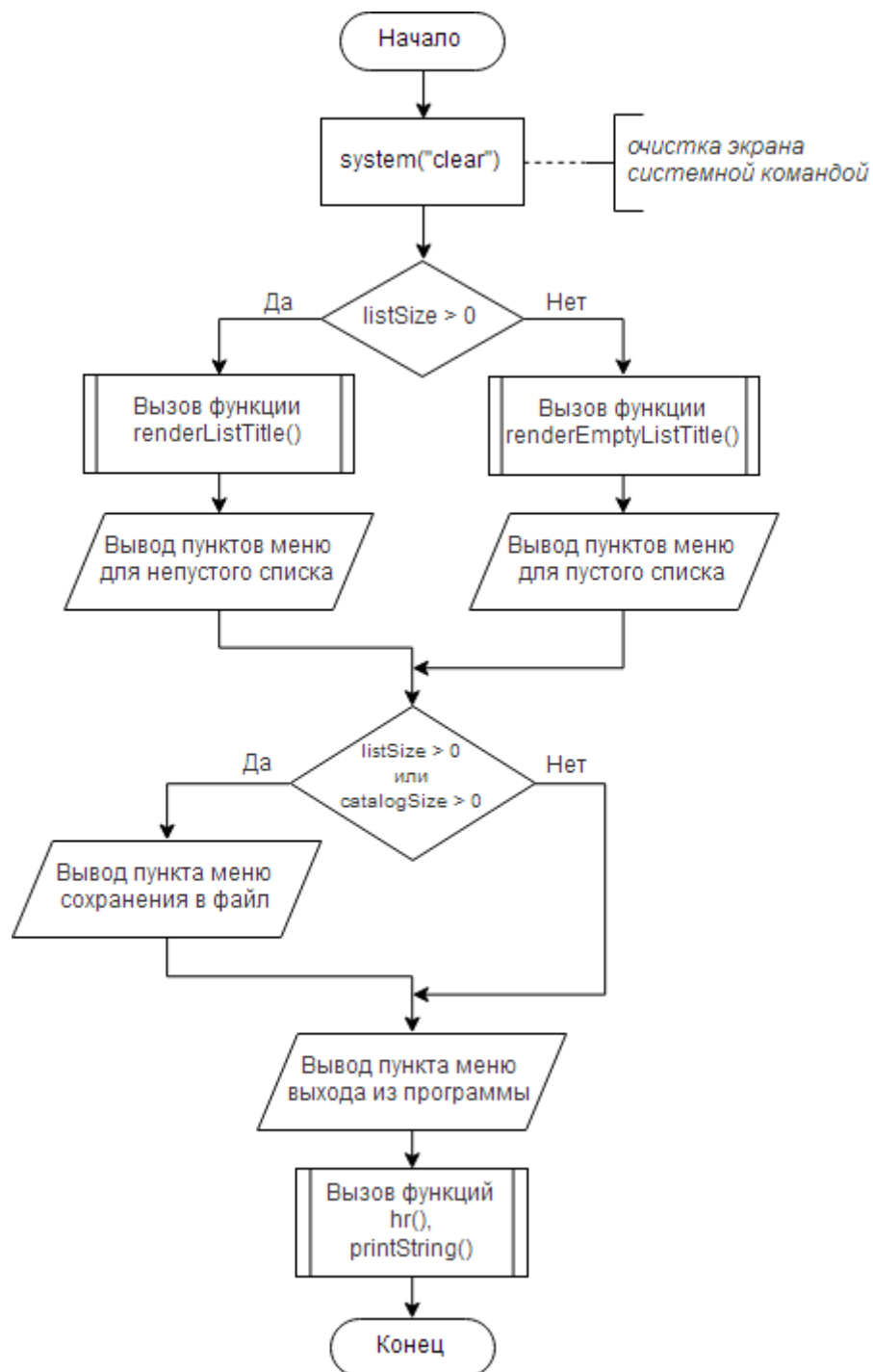


Рис. 4.3.7. Схема алгоритма функции *displayMainMenu()*

## 5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

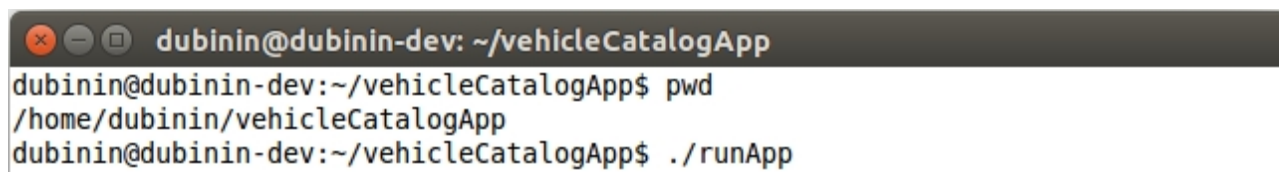
### 5. 1. Системные требования

Системные требования для использования программного средства минимальны, так как не требуется выполнять вычисления, которые затрачивают большие объемы оперативной памяти или используют большое количество квантов процессорного времени:

- архитектура процессора: x86 (Intel Pentium III и выше), x86-64;
- оперативная память: 128 Мб и выше;
- видеокарта, поддерживающая разрешение 800х600;
- монитор (CRT, LCD-TFT, LCD или любой другой), способный отображать выход на минимальное разрешение 800х600;
- клавиатура;
- операционная система семейства GNU/Linux.

### 5. 2. Инструкция пользователя

Для запуска программного средства каталога автомобилей необходимо из директории *vehicleCatalogApp* запустить исполняемый файл *runApp*, например, выполнив команду «./runApp» через консоль (рис. 5.2.1):



```
dubin@~$ cd ~/vehicleCatalogApp
dubin@~/vehicleCatalogApp$ pwd
/home/dubin/vehicleCatalogApp
dubin@~/vehicleCatalogApp$ ./runApp
```

Рис. 5.2.1. Команда запуска программного средства

После этого отобразится главное меню программы (рис. 5.2.2 — 5.2.3). При первом запуске программы будет создан бинарный файл «*vehicle\_catalog.dat*» в директории *vehicleCatalogApp*, который выполняет роль хранилища данных каталога автомобилей, и главное меню будет содержать только 1 пункт «Добавить автомобиль в каталог» (см. рис. 5.2.2), т. к. каталог еще пуст.

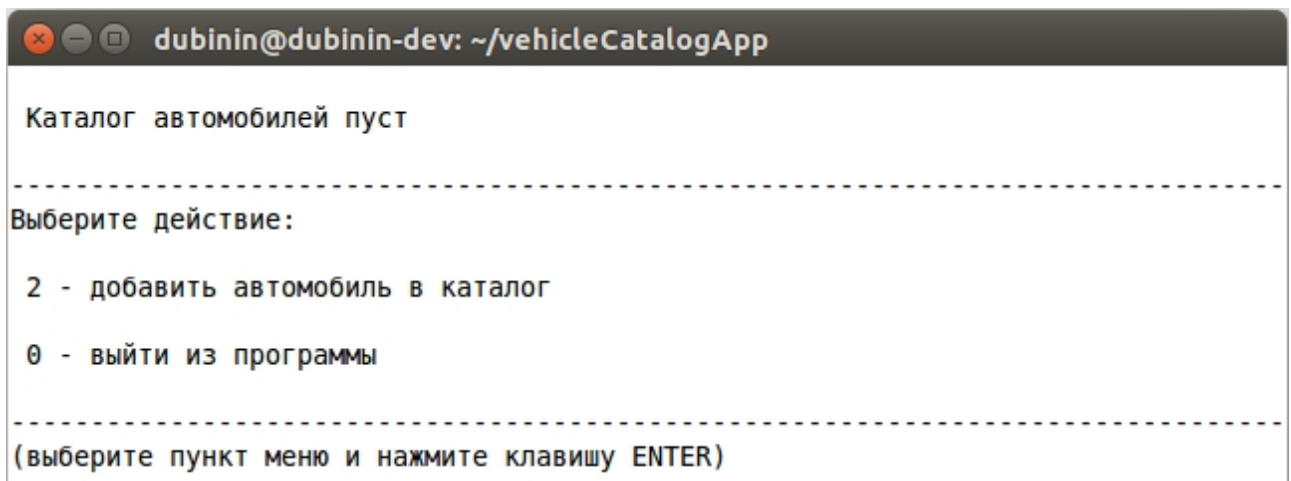


Рис. 5.2.2. Главное меню программы для пустого файла каталога



Рис. 5.2.3. Главное меню программы для непустого файла каталога

Для добавления нового автомобиля в каталог необходимо выбрать пункт меню #2 (нажать на клавиатуре цифру 2, затем ENTER). После этого отобразится интерфейс добавления автомобиля в каталог (рис. 5.2.4), где последовательно необходимо ввести следующие характеристики автомобиля:

- Марка автомобиля;
- Цена (в долларах США);
- Расход топлива (в количестве литров на 100 км.);
- Надежность (в количестве лет безотказной работы);
- Комфортность (в баллах).

После добавления автомобиля можно повторить процедуру добавления для другого автомобиля (нажать цифру 9, затем ENTER) или вернуться в главное меню программы (нажать цифру 0, затем ENTER).

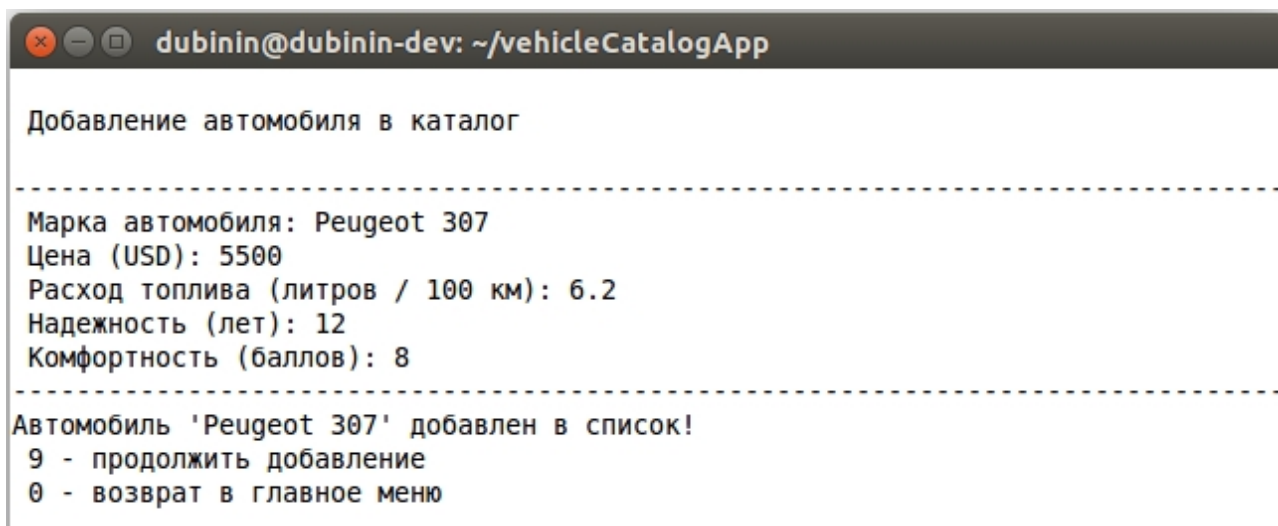


Рис. 5.2.4. Добавление автомобиля в каталог

Для отображения списка автомобилей в каталоге необходимо выбрать пункт меню #1 в главном меню программы (рис. 5.2.5).

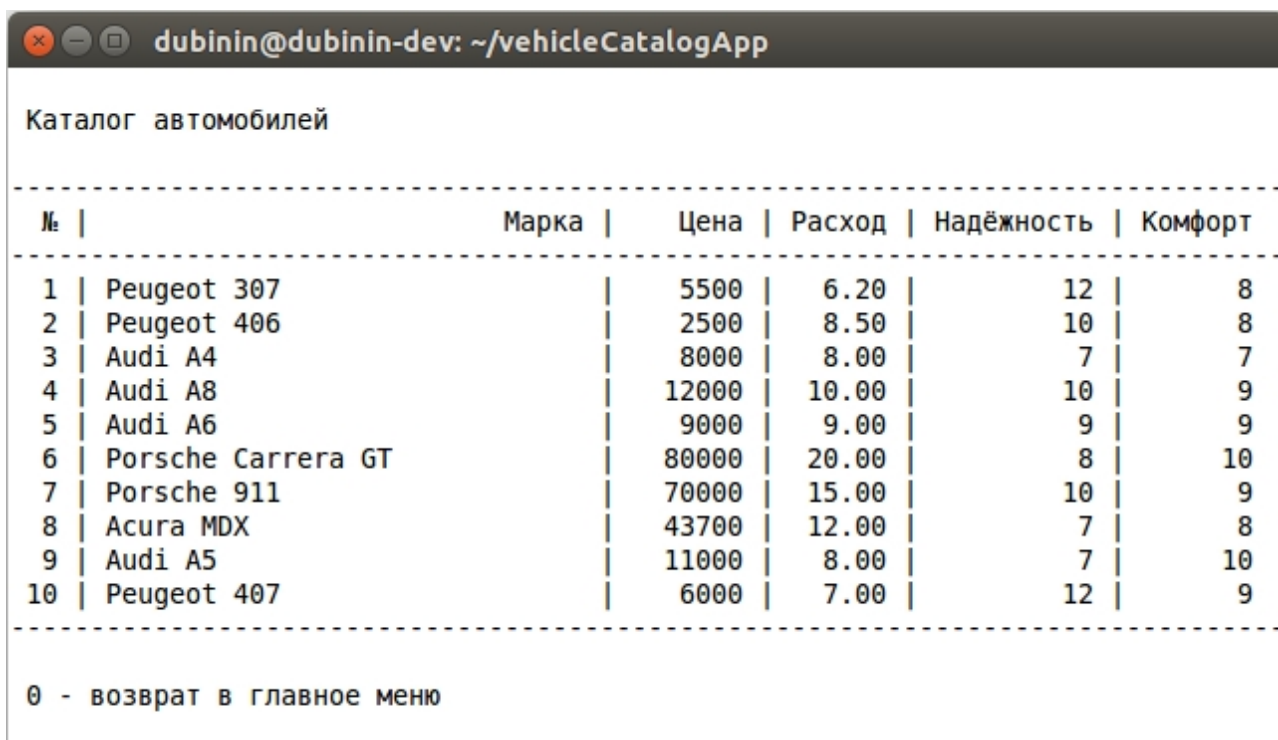


Рис. 5.2.5. Список автомобилей в каталоге

Для удаления автомобиля из каталога необходимо выбрать пункт меню #3 в главном меню программы, после чего отобразится список автомобилей с предложением удалить из него автомобиль по его номеру (рис. 5.2.6):



Удаление автомобиля из каталога

№	Марка	Цена	Расход	Надёжность	Комфорт
1	Peugeot 307	5500	6.20	12	8
2	Peugeot 406	2500	8.50	10	8
3	Audi A4	8000	8.00	7	7
4	Audi A8	12000	10.00	10	9
5	Audi A6	9000	9.00	9	9
6	Porsche Carrera GT	80000	20.00	8	10
7	Porsche 911	70000	15.00	10	9
8	Acura MDX	43700	12.00	7	8
9	Audi A5	11000	8.00	7	10
10	Peugeot 407	6000	7.00	12	9

Введите № из списка для удаления:

Рис. 5.2.6. Удаление автомобиля из каталога (интерфейс ввода номера)

Далее, выбрав номер автомобиля (например, №5) и нажав клавишу ENTER, указанный автомобиль будет удален из списка (рис. 5.2.7). После этого можно повторить процедуру удаления автомобиля (нажать цифру 9, затем ENTER) или вернуться в главное меню программы (нажать цифру 0, затем ENTER).

Для поиска автомобиля в каталоге в соответствии с требованиями покупателя необходимо выбрать пункт меню #4 в главном меню программы. Затем в открывшемся интерфейсе необходимо ввести характеристики автомобиля, указав их минимальное и максимальное значение (рис. 5.2.8). Если в каталоге найдены автомобили с заданными характеристиками, то будет отображен их список (рис. 5.2.9). Иначе, будет выведено сообщение о том, что в каталоге нет автомобилей с заданными характеристиками (рис. 5.2.10). После этого можно повторить процедуру поиска (нажать цифру 9, затем ENTER) или вернуться в главное меню программы (нажать цифру 0, затем ENTER).

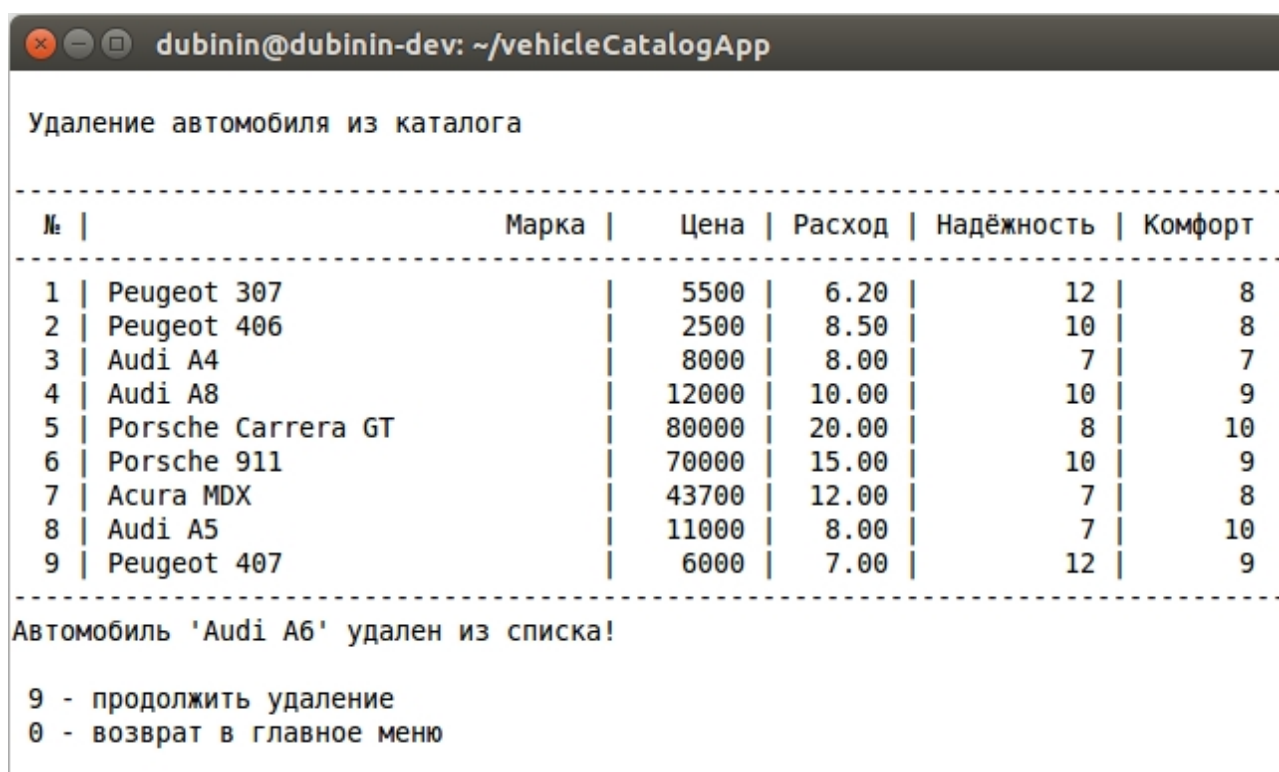


Рис. 5.2.7. Удаление автомобиля из каталога (интерфейс после удаления)

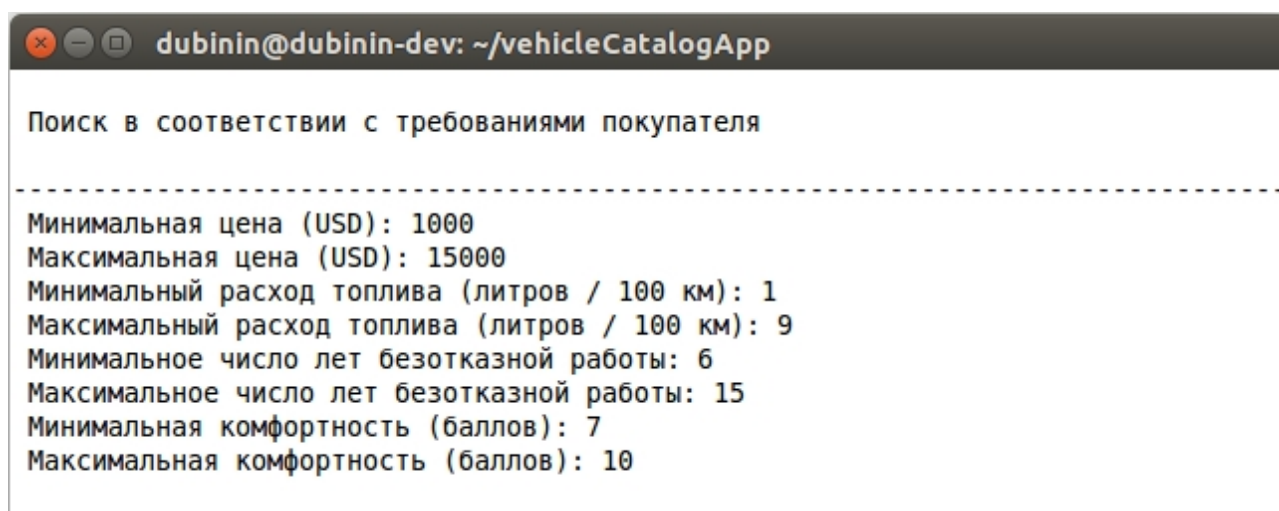


Рис. 5.2.8. Поиск в соответствии с требованиями покупателя (ввод значений)





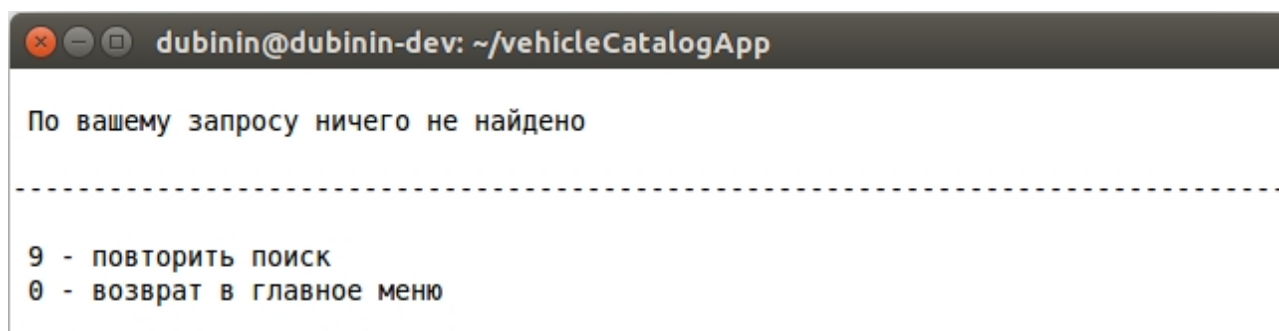
дubinin@dubinin-dev: ~/vehicleCatalogApp

Найденные в каталоге автомобили

№	Марка	Цена	Расход	Надёжность	Комфорт
1	Peugeot 307	5500	6.20	12	8
2	Peugeot 406	2500	8.50	10	8
3	Audi A4	8000	8.00	7	7
4	Audi A5	11000	8.00	7	10
5	Peugeot 407	6000	7.00	12	9

9 - повторить поиск  
0 - возврат в главное меню

Рис. 5.2.9. Поиск в соответствии с требованиями покупателя (результаты)



дubinin@dubinin-dev: ~/vehicleCatalogApp

По вашему запросу ничего не найдено

9 - повторить поиск  
0 - возврат в главное меню

Рис. 5.2.10. Поиск (не найдено)

Для поиска автомобиля в каталоге по соответствующему полю (марке автомобиля или его характеристике) необходимо выбрать пункт меню #5 в главном меню программы. После этого отобразится интерфейс выбора поля (рис. 5.2.11), где можно осуществить поиск автомобиля в каталоге по следующим полям:

- марка автомобиля;
- стоимость;
- расход топлива;
- надежность.

Для поиска по полю необходимо нажать цифру, соответствующую пункту меню, затем ENTER. Вернуться в главное меню программы можно, нажав цифру 0, затем ENTER.

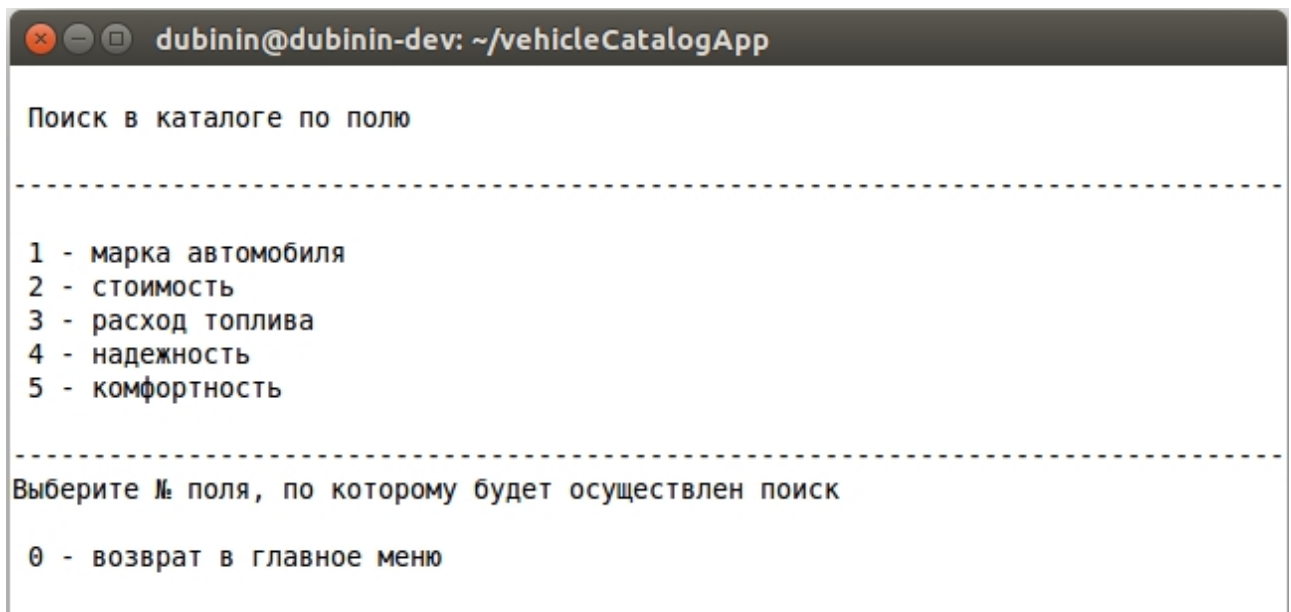


Рис. 5.2.11. Поиск по соответствующему полю (интерфейс выбора поля)

После выбора поля, по которому будет осуществлен поиск, будет выведен интерфейс ввода значения поля (рис. 5.2.12, рис. 5.2.14) и, после ввода значения и нажатия клавиши ENTER, результаты поиска по полю (рис. 5.2.13, рис. 5.2.15).

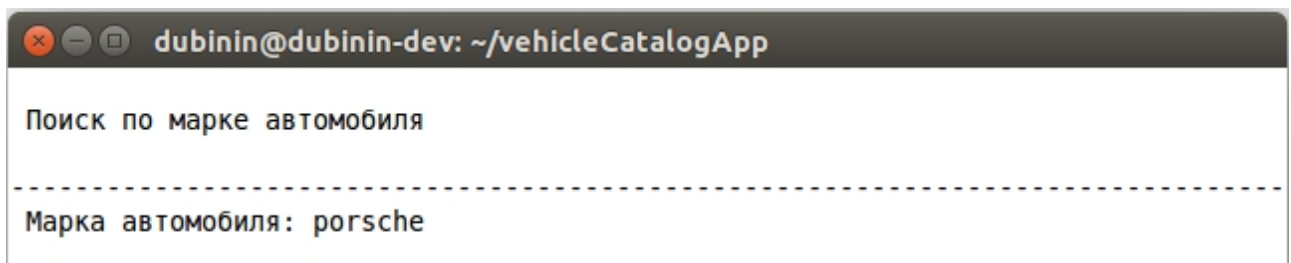


Рис. 5.2.12. Поиск по полю (интерфейс ввода значения)

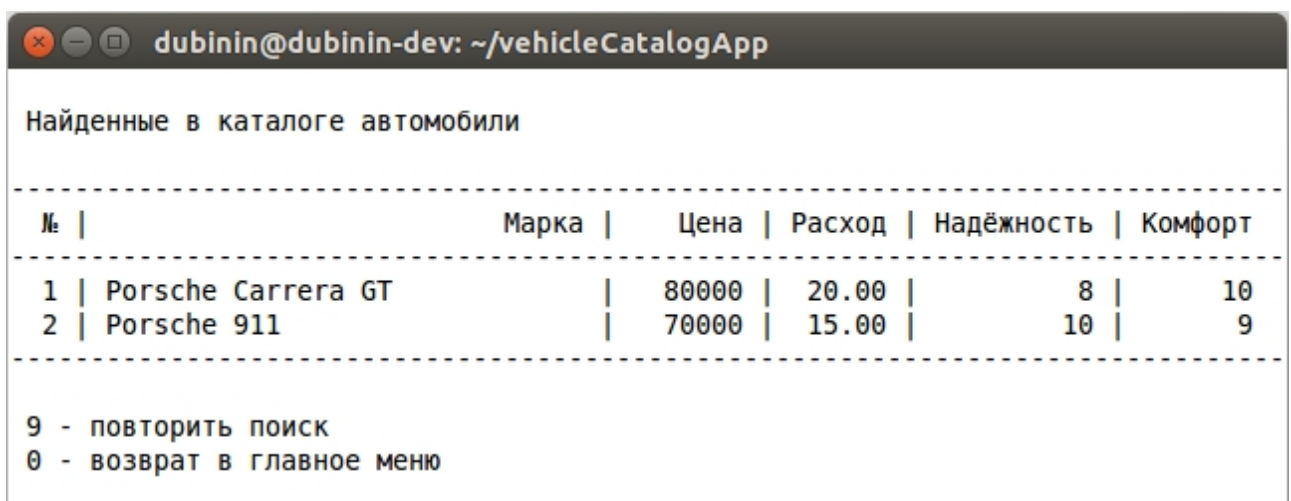


Рис. 5.2.13. Поиск по полю (результаты)



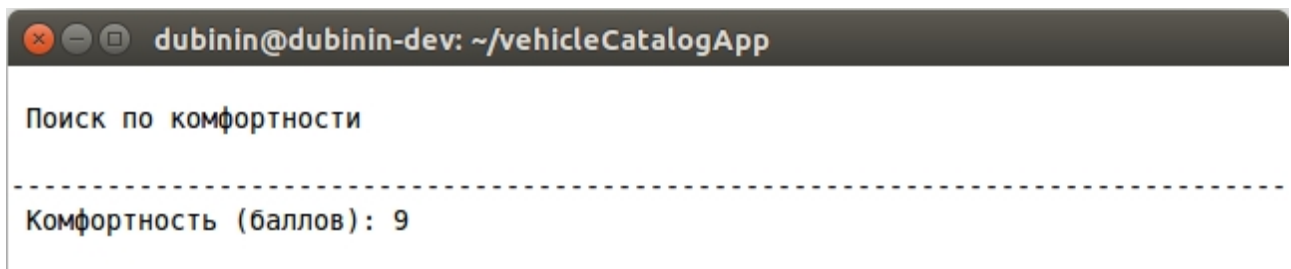


Рис. 5.2.14. Поиск по полю (интерфейс ввода значения)

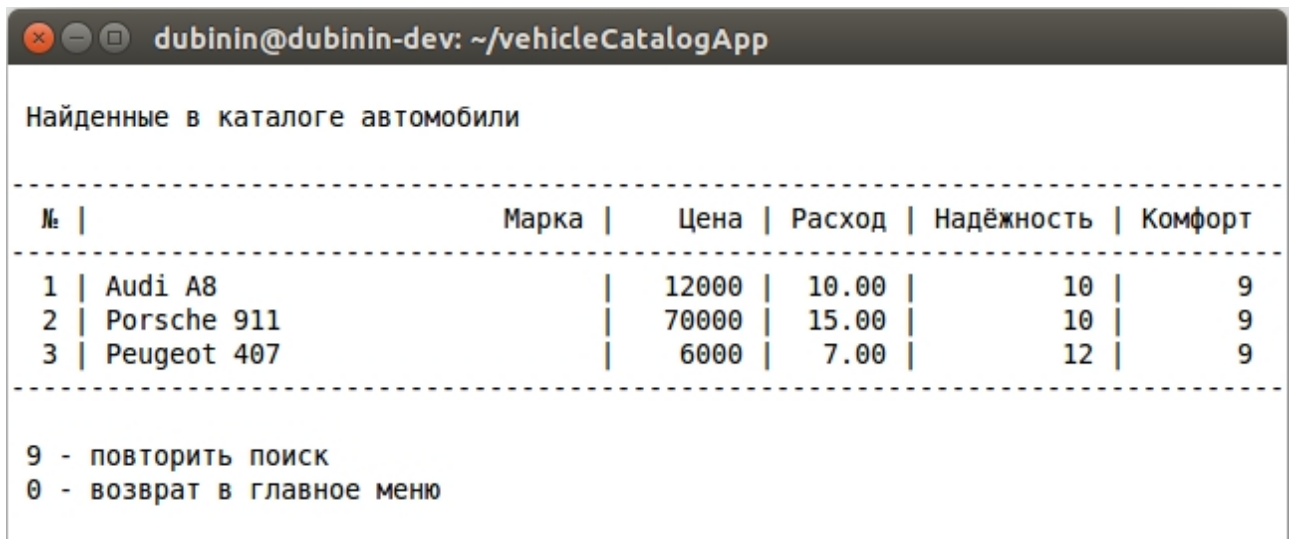


Рис. 5.2.15. Поиск по полю (результаты)

Для сортировки списка автомобилей в каталоге по алфавиту необходимо выбрать пункт меню #6 в главном меню программы. После этого отобразится сообщение об успешной сортировке (рис. 5.2.16).

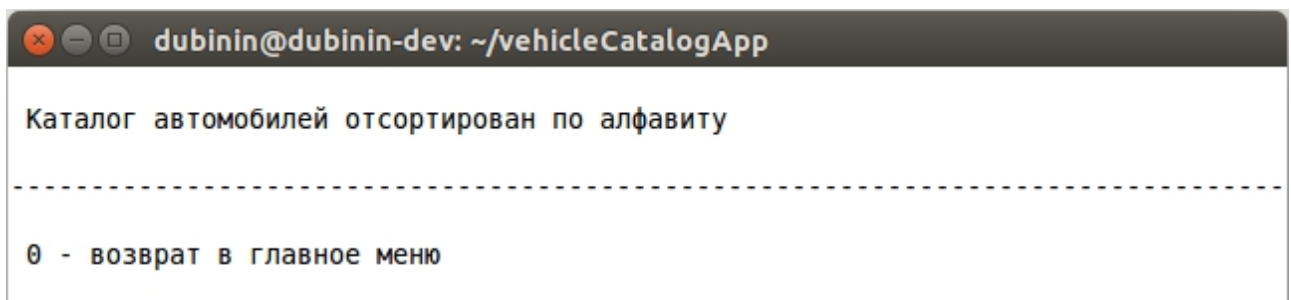


Рис. 5.2.16. Сортировка списка автомобилей по алфавиту

Убедиться в корректности сортировки можно, отобразив список автомобилей (пункт #1 в главном меню программы):

№	Марка	Цена	Расход	Надёжность	Комфорт
1	Acura MDX	43700	12.00	7	8
2	Audi A4	8000	8.00	7	7
3	Audi A5	11000	8.00	7	10
4	Audi A8	12000	10.00	10	9
5	Peugeot 307	5500	6.20	12	8
6	Peugeot 406	2500	8.50	10	8
7	Peugeot 407	6000	7.00	12	9
8	Porsche 911	70000	15.00	10	9
9	Porsche Carrera GT	80000	20.00	8	10

0 - возврат в главное меню

Рис. 5.2.17. Отсортированный список автомобилей

После всех изменений в каталоге автомобилей необходимо сохранить обновленные данные в файл-хранилище «*vehicle\_catalog.dat*». Для этого нужно выбрать пункт меню #9 в главном меню программы (рис. 5.2.18).

Изменения в каталоге успешно сохранены
----------------------------------------

0 - возврат в главное меню

Рис. 5.2.18. Сохранение изменений в файле каталога

Это необходимо выполнять для сохранения данных, так как все изменения (добавление, удаление, сортировка), производимые во время работы одной сессии программы, осуществляются в оперативной памяти и после выхода из программы будут утеряны.

## 6. СПИСОК ЛИТЕРАТУРЫ

- [1] Макконелл С. Совершенный код. Мастер-класс. — М.: Издательство «Русская редакция», 2017. — 896 с.
- [2] Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ. — СПб, 2003.
- [3] Себеста Р. Основные концепции языков программирования. — М.: Издательский дом «Вильямс», 2001, 672 с.
- [4] Керниган Б., Пайк Р. Практика программирования. — М.: Издательский дом «Вильямс», 2004, 288 с.
- [5] Бентли. Дж. Жемчужины программирования. — СПб.: Питер, 2002, 272 с.
- [6] Керниган Б., Ричи Д. Язык программирования С. — М.: Издательский дом «Вильямс», 2007, 304 с.
- [7] Харибсон С., Стил Г. Язык программирования С. — М.: СПб.: БХВ-Петербург, 2008, 896 с.
- [8] ГОСТ 19.701-90 — Единая система программной документации — Схемы алгоритмов, программ, данных и систем – Условные обозначения и правила выполнения.
- [9] Глухова Л. А. Электронный учебно-методический комплекс «Основы алгоритмизации и программирования»: Учеб. пособие. Часть 1. — Мн.: БГУИР, 2010.
- [10] Глухова Л. А. Электронный учебно-методический комплекс «Основы алгоритмизации и программирования»: Учеб. пособие. Часть 2. — Мн.: БГУИР, 2010.
- [11] Л. В. Серебряная. Электронный учебно-методический комплекс «Структуры и алгоритмы обработки данных» — Мн.: БГУИР, 2010.
- [12] В. В. Бахтизин, Д.Е. Оношко. Электронный ресурс учебной дисциплины «Языки программирования (часть 1)» — Мн.: БГУИР, 2016.
- [13] Бахтизин В. В., Марина И. М., Шостак Е. В. Учебно-методическое пособие по курсу «Конструирование программ и языки программирования». Ч. 1. — Мн: Ротапринт БГУИР, 2006.
- [14] C standard library — Wikipedia [Электронный ресурс]. — Электронные данные. — Режим доступа: [https://en.wikipedia.org/wiki/C\\_standard\\_library](https://en.wikipedia.org/wiki/C_standard_library)
- [15] C reference [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://en.cppreference.com/w/c>
- [16] Linux man pages online [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://man7.org/linux/man-pages/index.html>

- [17] Bubble sort — Wikipedia [Электронный ресурс]. — Электронные данные. — Режим доступа: [https://en.wikipedia.org/wiki/Bubble\\_sort](https://en.wikipedia.org/wiki/Bubble_sort)
- [18] Linked list — Wikipedia [Электронный ресурс]. — Электронные данные. — Режим доступа: [https://en.wikipedia.org/wiki/Linked\\_list](https://en.wikipedia.org/wiki/Linked_list)

## 7. ЛИСТИНГ ПРОГРАММЫ

### 7. 1. Листинг исходного кода модуля *main*

```
/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

// Подключение заголовочного файла
#include "application/runner.h"

void main()
{
    runApplication();
    return;
}
```

### 7. 2. Листинг исходных кодов модулей пакета *application*

#### 7. 2. 1. Листинг исходного кода заголовочного файла *definitions.h*

```
/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

#ifndef DEFINITIONS_H
#define DEFINITIONS_H

// Объявление макроса (перевод курсора на новую строку)
#define BR putchar('\n')

// Объявление символических констант
#define CALL_CLEAR          "clear"
#define CATALOG_FILE        "vehicle_catalog.dat"
#define CATALOG_READ_MODE   "a+b"
#define CATALOG_WRITE_MODE "wb"
#define WINDOW_WIDTH        79
#define MODEL_NAME_SIZE     50
#define NULL_TERMINATED     '\0'
#define HYPHEN               '-'
#define ITEM_0               '0'
#define ITEM_1               '1'
#define ITEM_2               '2'
#define ITEM_3               '3'
#define ITEM_4               '4'
#define ITEM_5               '5'
#define ITEM_6               '6'
#define ITEM_9               '9'
```

```

// Объявление символических констант (сообщения пользовательского интерфейса)
#define MSG_CATALOG_TITLE "Каталог автомобилей"
#define MSG_EMPTY_CATALOG_TITLE "Каталог автомобилей пуст"
#define MSG_ADD_CAR_TITLE "Добавление автомобиля в каталог"
#define MSG_REMOVE_CAR_TITLE "Удаление автомобиля из каталога"
#define MSG_BY_WISHES_TITLE "Поиск в соответствии с требованиями покупателя"
#define MSG_BY_FIELD_TITLE "Поиск в каталоге по полю"
#define MSG_BY_MODEL_TITLE "Поиск по марке автомобиля"
#define MSG_BY_PRICE_TITLE "Поиск по стоимости"
#define MSG_BY_FUEL_TITLE "Поиск по расходу топлива"
#define MSG_BY_QUALITY_TITLE "Поиск по надежности"
#define MSG_BY_COMFORT_TITLE "Поиск по комфортности"
#define MSG_FOUND_TITLE "Найденные в каталоге автомобили"
#define MSG_NOT_FOUND "По вашему запросу ничего не найдено"
#define MSG_SORTED "Каталог автомобилей отсортирован по алфавиту"
#define MSG_SAVED "Изменения в каталоге успешно сохранены"
#define MSG_MEMORY_ERR "Ошибка выделения памяти элементу списка"
#define MSG_SELECT_ACTION "Выберите действие:"
#define MSG_DISPLAY_CATALOG " 1 - вывести каталог на экран"
#define MSG_ADD_CAR " 2 - добавить автомобиль в каталог"
#define MSG_REMOVE_CAR " 3 - удалить автомобиль из каталога"
#define MSG_SEARCH_BY_WISHES " 4 - поиск в каталоге по требованию"
#define MSG_SEARCH_BY_FIELD " 5 - поиск в каталоге по полю"
#define MSG_SORT_BY_ALPHABET " 6 - сортировка каталога по алфавиту"
#define MSG_SAVE_CATALOG " 9 - сохранить изменения в каталоге"
#define MSG_APP_EXIT " 0 - выйти из программы"
#define MSG_DO_SELECT "(выберите пункт меню и нажмите клавишу ENTER) "
#define MSG_MODEL " Марка автомобиля: "
#define MSG_PRICE " Цена (USD): "
#define MSG_FUEL " Расход топлива (литров / 100 км): "
#define MSG_QUALITY " Надежность (лет): "
#define MSG_COMFORT " Комфортность (баллов): "
#define MSG_ADD_SUCCESSFUL "Автомобиль '%s' добавлен в список!\n"
#define MSG_ADD_CONTINUE " 9 - продолжить добавление"
#define MSG_REMOVE_SELECT "Введите № из списка для удаления: "
#define MSG_REMOVE_SUCCESSFUL "Автомобиль '%s' удален из списка!\n"
#define MSG_REMOVE_CONTINUE " 9 - продолжить удаление"
#define MSG_SEARCH_PRICE_MIN " Минимальная цена (USD): "
#define MSG_SEARCH_PRICE_MAX " Максимальная цена (USD): "
#define MSG_SEARCH_FUEL_MIN " Минимальный расход топлива (литров / 100 км): "
#define MSG_SEARCH_FUEL_MAX " Максимальный расход топлива (литров / 100 км): "
#define MSG_SEARCH_QUALITY_MIN " Минимальное число лет безотказной работы: "
#define MSG_SEARCH_QUALITY_MAX " Максимальное число лет безотказной работы: "
#define MSG_SEARCH_COMFORT_MIN " Минимальная комфортность (баллов): "
#define MSG_SEARCH_COMFORT_MAX " Максимальная комфортность (баллов): "
#define MSG_SEARCH_BY_MODEL " 1 - марка автомобиля"
#define MSG_SEARCH_BY_PRICE " 2 - стоимость"
#define MSG_SEARCH_BY_FUEL " 3 - расход топлива"
#define MSG_SEARCH_BY_QUALITY " 4 - надежность"
#define MSG_SEARCH_BY_COMFORT " 5 - комфортность"

```

```

#define MSG_SEARCH_SELECT      "Выберите № поля, по которому будет осуществлен
поиск"
#define MSG_SEARCH_REPEAT      " 9 - повторить поиск"
#define MSG_RETURN_TO_MENU     " 0 - возврат в главное меню "

#define COLUMN_NUMBER  "№ | "
#define COLUMN_MODEL    "Марка | "
#define COLUMN_PRICE    "Цена | "
#define COLUMN_FUEL     "Расход | "
#define COLUMN_QUALITY  "Надёжность | "
#define COLUMN_COMFORT  "Комфорт"

#define PATTERN_TITLE      " %s\n\n"
#define PATTERN_LIST_ROW   "%3u | %-30s | %7lu | %6.2f | %10u | %7u\n"
#define PATTERN_LIST_HEADER "%8s%38s%14s%s%s\n"
#define PATTERN_FULL_STR   "%[^\\n]s"
#define PATTERN_STR        "%s"
#define PATTERN_UNSIGNED_LONG "%lu"
#define PATTERN_FLOAT       "%f"
#define PATTERN_UNSIGNED    "%u"

#endif

```

## 7. 2. 2. Листинг исходного кода заголовочного файла *types.h*

```

/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

#ifndef TYPES_H
#define TYPES_H

// Объявление новой структуры и имени типа - vehicle
typedef struct {
    char          model[MODEL_NAME_SIZE]; // марка автомобиля
    unsigned long int price;                // цена, в USD
    float          fuelConsumption;         // расход топлива, в литрах/100км
    unsigned int   longTermQuality;         // надёжность, в годах
    unsigned int   comfort;                 // комфортность, в баллах
} vehicle;

/**
 * Объявление новой структуры node, описывающей элемент списка автомобилей,
 * и имени типа list
 */
typedef struct node {
    vehicle car;
    struct node *next;
} list;

// Объявление нового имени типа listPtr - указатель на список автомобилей
typedef list *listPtr;

// Объявление нового имени типа actionType - указатель на функцию
typedef void (*actionType) (listPtr *index);

```

```
#endif
```

### 7. 2. 3. Листинг исходного кода заголовочного файла *runner.h*

```
/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

#ifndef RUNNER_H
#define RUNNER_H

/*
 * Прототип функции
 */
void runApplication();

#endif
```

### 7. 2. 4. Листинг исходного кода файла реализации *runner.c*

```
/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

// Подключение заголовочных файлов
#include <stdio.h>
#include <stdlib.h>
#include "definitions.h"
#include "types.h"
#include "runner.h"
#include "../handler/handler.h"

/**
 * Функция запуска и исполнения программы.
 */
void runApplication()
{
    /*
     * catalogSize - количество элементов файла каталога,
     * listSize - количество элементов динамического списка,
     * isExit - флаг выхода из программы
     */
    unsigned int catalogSize, listSize, isExit;
    char actionKey;

    /*
     * Открытие файла каталога, формирование динамического списка на основе
     * данных из файла, закрытие файла каталога.
     */
    FILE *catalog = fopen(CATALOG_FILE, CATALOG_READ_MODE);
    listPtr index = formList(catalog);
    fclose(catalog);
```



```

// Вычисление количества элементов из файла каталога
catalogSize = countListSize(&index);

isExit = 0;
do {
    // Вычисление количества элементов динамического списка
    listSize = countListSize(&index);

    displayMainMenu(listSize, catalogSize);
    actionKey = getchar();
    switch (actionKey) {

        // Процедура отображения списка автомобилей
        case ITEM_1:
            if (listSize > 0) {
                renderListTitle();
                displayCarsList(&index);
                returnToMainMenu();
            }
            break;

        // Процедура добавления элемента в список
        case ITEM_2:
            doRepeatedAction(&index, addCarsIntoList);
            break;

        // Процедура удаление элемента из списка
        case ITEM_3:
            if (listSize > 0) {
                doRepeatedAction(&index, removeCarsFromList);
            }
            break;

        // Процедура поиска в соответствии с требованиями покупателя
        case ITEM_4:
            if (listSize > 0) {
                doRepeatedAction(&index, searchByWishes);
            }
            break;

        // Процедура поиска по соответствующему полю
        case ITEM_5:
            if (listSize > 0) {
                searchByField(&index);
            }
            break;

        // Процедура сортировки по алфавиту
        case ITEM_6:
            if (listSize > 0) {
                sortByAlphabet(&index);
                returnToMainMenu();
            }
            break;

        // Процедура записи в файл элементов динамического списка
        case ITEM_9:
            if ((listSize > 0) || (catalogSize > 0)) {
                catalogSize = saveCatalog(&index);
                returnToMainMenu();
            }
            break;
    }
}

```

```

        // Выход из программы
        case ITEM_0:
            isExit = 1;
            break;
    }
} while (isExit == 0);

free(index);
return;
}

```

### 7. 3. Листинг исходного кода модуля *handler*

#### 7. 3. 1. Листинг исходного кода заголовочного файла *handler.h*

```

/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

#ifndef HANDLER_H
#define HANDLER_H

/*
 * Прототипы функций
 */
void doRepeatedAction(listPtr *index, actionType action);
listPtr formList(FILE *catalog);
unsigned int countListSize(listPtr *index);
void displayCarsList(listPtr *index);
void addCarsIntoList(listPtr *index);
void removeCarsFromList(listPtr *index);
void searchByWishes(listPtr *index);
void searchByField(listPtr *index);
void sortByAlphabet(listPtr *index);
unsigned int saveCatalog(listPtr *index);

#endif

```

#### 7. 3. 2. Листинг исходного кода файла реализации *handler.c*

```

/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

// Подключение заголовочных файлов
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../application/definitions.h"
#include "../application/types.h"
#include "handler.h"

/**

```

```

* Функция вызова повторяющихся функций программы.
* index - параметр типа listPtr (указатель на заглавный элемент списка)
* action - параметр типа actionType (указатель на функцию)
*/
void doRepeatedAction(listPtr *index, actionType action)
{
    char actionKey;
    action(index);
    do {
        actionKey = getchar();
        if (actionKey == ITEM_9) {
            action(index);
        }
    } while (actionKey != ITEM_0);
    return;
}

/**
* Функция, формирующая динамический список автомобилей из файловой переменной.
* Возвращает линейный однонаправленный список.
* catalog - параметр файлового типа
*/
listPtr formList(FILE *catalog)
{
    // Указатели на заглавное и текущее звено списка
    listPtr index, current;
    vehicle car;

    // Выделение памяти с проверкой заглавному элементу списка
    index = (list *) malloc(sizeof(list));
    if (index == NULL) {
        puts(MSG_MEMORY_ERR);
        return NULL;
    }

    current = index;
    current->next = NULL;

    /*
    * Чтение элементов типизированного файла
    * и формирование динамического списка из этих элементов
    */
    while (1) {
        if (fread(&car, sizeof(vehicle), 1, catalog) < 1) {
            break;
        }

        // Выделение памяти с проверкой новому элементу списка
        current->next = (list *) malloc(sizeof(list));
        if (current->next == NULL) {
            puts(MSG_MEMORY_ERR);
            return NULL;
        }

        current = current->next;
        current->car = car;
        current->next = NULL;
    }

    return index;
}

```

```

/**
 * Функция, возвращающая количество элементов списка.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
unsigned int countListSize(listPtr *index)
{
    listPtr current;
    unsigned int listSize;

    current = (*index)->next;
    listSize = 0;
    while (current != NULL) {
        listSize++;
        current = current->next;
    }
    return listSize;
}

/**
 * Функция вывода на экран списка автомобилей.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
void displayCarsList(listPtr *index)
{
    renderCarsListHeader();

    listPtr current = (*index)->next;
    unsigned int i = 0;
    while (current != NULL) {
        i++;
        renderCarsListRow(i, current->car);
        current = current->next;
    }
    hr();
    return;
}

/**
 * Функция добавление нового элемента в список автомобилей.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
void addCarsIntoList(listPtr *index)
{
    vehicle newCar;
    listPtr current, newNode;

    renderAddCarTitle();
    getchar();
    printString(MSG_MODEL);
    scanf(PATTERN_FULL_STR, newCar.model);
    printString(MSG_PRICE);
    scanf(PATTERN_UNSIGNED_LONG, &newCar.price);
    printString(MSG_FUEL);
    scanf(PATTERN_FLOAT, &newCar.fuelConsumption);
    printString(MSG_QUALITY);
    scanf(PATTERN_UNSIGNED, &newCar.longTermQuality);
    printString(MSG_COMFORT);
    scanf(PATTERN_UNSIGNED, &newCar.comfort);

    current = *index;
    while (current->next != NULL) {
        current = current->next;
    }
}

```

```

    }

    newNode = (list *) malloc(sizeof(list));
    if (newNode == NULL) {
        puts(MSG_MEMORY_ERR);
        return;
    }
    newNode->car = newCar;
    newNode->next = current->next;
    current->next = newNode;
    hr();
    printf(MSG_ADD_SUCCESSFUL, newCar.model);
    puts(MSG_ADD_CONTINUE);
    puts(MSG_RETURN_TO_MENU);
    return;
}

/**
 * Функция удаления элемента из списка автомобилей.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
void removeCarsFromList(listPtr *index)
{
    listPtr current, removed;
    unsigned int listSize, i, n;

    listSize = countListSize(index);
    if (listSize > 0) {
        renderRemoveCarTitle();
        displayCarsList(index);
        current = *index;
        printString(MSG_REMOVE_SELECT);
        scanf(PATTERN_UNSIGNED, &n);

        // Проверка на существование введенного № элемента
        if (n > 0 && n <= listSize) {
            for (i = 1; i < n; i++) {
                current = current->next;
            }
            removed = current->next;
            current->next = removed->next;
            renderRemoveCarTitle();
            displayCarsList(index);
            printf(MSG_REMOVE_SUCCESSFUL, removed->car.model);
            free(removed);
        }
        BR;
        puts(MSG_REMOVE_CONTINUE);
    } else {
        renderEmptyListTitle();
        BR;
    }
    puts(MSG_RETURN_TO_MENU);
    return;
}

/**
 * Функция поиска автомобилей в соответствии с требованиями покупателя.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
void searchByWishes(listPtr *index)
{

```

```

listPtr current;
vehicle car;
unsigned int i;
unsigned long int minPrice, maxPrice;
float minFuelConsumption, maxFuelConsumption;
unsigned int minLongTermQuality, maxLongTermQuality;
unsigned int minComfort, maxComfort;

renderSearchByWishesTitle();
printString(MSG_SEARCH_PRICE_MIN);
scanf(PATTERN_UNSIGNED_LONG, &minPrice);
printString(MSG_SEARCH_PRICE_MAX);
scanf(PATTERN_UNSIGNED_LONG, &maxPrice);
printString(MSG_SEARCH_FUEL_MIN);
scanf(PATTERN_FLOAT, &minFuelConsumption);
printString(MSG_SEARCH_FUEL_MAX);
scanf(PATTERN_FLOAT, &maxFuelConsumption);
printString(MSG_SEARCH_QUALITY_MIN);
scanf(PATTERN_UNSIGNED, &minLongTermQuality);
printString(MSG_SEARCH_QUALITY_MAX);
scanf(PATTERN_UNSIGNED, &maxLongTermQuality);
printString(MSG_SEARCH_COMFORT_MIN);
scanf(PATTERN_UNSIGNED, &minComfort);
printString(MSG_SEARCH_COMFORT_MAX);
scanf(PATTERN_UNSIGNED, &maxComfort);

i = 0;
current = *index;
while (current->next != NULL) {
    car = current->next->car;

    /*
     * Проверка на вхождение характеристик автомобиля из списка,
     * в диапазон заданных покупателем значений.
     */
    if ((car.price >= minPrice)
        && (car.price <= maxPrice)
        && (car.fuelConsumption >= minFuelConsumption)
        && (car.fuelConsumption <= maxFuelConsumption)
        && (car.longTermQuality >= minLongTermQuality)
        && (car.longTermQuality <= maxLongTermQuality)
        && (car.comfort >= minComfort)
        && (car.comfort <= maxComfort)
    ) {
        i++;
        renderSearchResult(i, car);
    }
    current = current->next;
}
renderSearchResultFooter(i);
return;
}

/**
 * Функция поиска автомобилей по соответствующему полю.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
void searchByField(listPtr *index)
{
    listPtr current;
    vehicle car;
    unsigned int i, isRepeat, isExit;

```

```

char actionKey;
char specificModel[MODEL_NAME_SIZE];
char normalizedSpecificModel[MODEL_NAME_SIZE];
char normalizedModel[MODEL_NAME_SIZE];
unsigned long int specificPrice;
float specificFuelConsumption;
unsigned int specificLongTermQuality;
unsigned int specificComfort;

renderSearchByFieldTitle();
BR;
puts(MSG_SEARCH_BY_MODEL);
puts(MSG_SEARCH_BY_PRICE);
puts(MSG_SEARCH_BY_FUEL);
puts(MSG_SEARCH_BY_QUALITY);
puts(MSG_SEARCH_BY_COMFORT);
BR;
hr();
puts(MSG_SEARCH_SELECT);
BR;
puts(MSG_RETURN_TO_MENU);

do {
    i = 0;
    current = *index;
    isRepeat = 0;
    isExit = 0;
    actionKey = getchar();
    switch (actionKey) {
        case ITEM_1:

            /*
             * Поиск по марке автомобиля реализован путем нахождения
             * вхождения введенной строки в наименовании марки,
             * без учета регистра.
             */
            renderSearchByModelTitle();
            getchar();
            printString(MSG_MODEL);
            scanf(PATTERN_FULL_STR, specificModel);
            while (current->next != NULL) {
                car = current->next->car;
                strcpy(normalizedSpecificModel, specificModel);
                toUpperCase(normalizedSpecificModel);
                strcpy(normalizedModel, car.model);
                toUpperCase(normalizedModel);
                if (strstr(normalizedModel, normalizedSpecificModel) !=
NULL) {
                    i++;
                    renderSearchResult(i, car);
                }
                current = current->next;
            }
            renderSearchResultFooter(i);
            break;
        case ITEM_2:
            renderSearchByPriceTitle();
            printString(MSG_PRICE);
            scanf(PATTERN_UNSIGNED_LONG, &specificPrice);
            while (current->next != NULL) {
                car = current->next->car;
                if (specificPrice == car.price) {

```

```

        i++;
        renderSearchResult(i, car);
    }
    current = current->next;
}
renderSearchResultFooter(i);
break;
case ITEM_3:
    renderSearchByFuelConsumptionTitle();
    printString(MSG_FUEL);
    scanf(PATTERN_FLOAT, &specificFuelConsumption);
    while (current->next != NULL) {
        car = current->next->car;
        if (specificFuelConsumption == car.fuelConsumption) {
            i++;
            renderSearchResult(i, car);
        }
        current = current->next;
    }
    renderSearchResultFooter(i);
    break;
case ITEM_4:
    renderSearchByLongTermQualityTitle();
    printString(MSG_QUALITY);
    scanf(PATTERN_UNSIGNED, &specificLongTermQuality);
    while (current->next != NULL) {
        car = current->next->car;
        if (specificLongTermQuality == car.longTermQuality) {
            i++;
            renderSearchResult(i, car);
        }
        current = current->next;
    }
    renderSearchResultFooter(i);
    break;
case ITEM_5:
    renderSearchByComfortTitle();
    printString(MSG_COMFORT);
    scanf(PATTERN_UNSIGNED, &specificComfort);
    while (current->next != NULL) {
        car = current->next->car;
        if (specificComfort == car.comfort) {
            i++;
            renderSearchResult(i, car);
        }
        current = current->next;
    }
    renderSearchResultFooter(i);
    break;
case ITEM_9:
    isRepeat = 1;
    isExit = 1;
    break;
case ITEM_0:
    isExit = 1;
    break;
}
} while (isExit == 0);

if (isRepeat == 1) {
    searchByField(index);
}

```



```

        return;
    }

/**
 * Функция сортировки по алфавиту списка по марке автомобиля.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
void sortByAlphabet(listPtr *index)
{
    listPtr current;
    vehicle car;
    unsigned int isSorted;
    char normalizedCurrentModel[MODEL_NAME_SIZE];
    char normalizedNextModel[MODEL_NAME_SIZE];

    do {
        isSorted = 1;
        current = (*index)->next;
        while (current->next != NULL) {

            /*
             * Сортировка без учета регистра.
             * Для этого сравниваются марки автомобилей, приведенные
             * к верхнему регистру.
             */
            strcpy(normalizedCurrentModel, current->car.model);
            toUpperCase(normalizedCurrentModel);
            strcpy(normalizedNextModel, current->next->car.model);
            toUpperCase(normalizedNextModel);
            if (strcmp(normalizedCurrentModel, normalizedNextModel) > 0) {
                car = current->car;
                current->car = current->next->car;
                current->next->car = car;
                isSorted = 0;
            }
            current = current->next;
        }
    } while (isSorted == 0);

    renderSortedMsg();
    return;
}

/**
 * Функция для сохранения информации об автомобилях из динамического списка
 * в файл каталога.
 * Возвращает количество элементов списка.
 * index - параметр типа listPtr (указатель на заглавный элемент списка)
 */
unsigned int saveCatalog(listPtr *index)
{
    FILE *catalog;
    listPtr current;
    unsigned int catalogSize;

    catalog = fopen(CATALOG_FILE, CATALOG_WRITE_MODE);
    current = (*index)->next;
    catalogSize = 0;
    while (current != NULL) {
        fwrite(&current->car, sizeof(vehicle), 1, catalog);
        current = current->next;
        catalogSize++;
    }
}

```

```

    }
    fclose(catalog);

    renderSavedMsg();
    return catalogSize;
}

```

## 7. 4. Листинг исходного кода модуля *view*

### 7. 4. 1. Листинг исходного кода заголовочного файла *view.h*

```

/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.
 * 08.03.2017
 */

#ifndef VIEW_H
#define VIEW_H

/*
 * Прототипы функций
 */
void hr();
void toUpCase(char *str);
void printString(const char *str);
void renderTitle(const char *title);
void renderListTitle();
void renderEmptyListTitle();
void renderAddCarTitle();
void renderRemoveCarTitle();
void renderSearchByWishesTitle();
void renderSearchByFieldTitle();
void renderSearchByModelTitle();
void renderSearchByPriceTitle();
void renderSearchByFuelConsumptionTitle();
void renderSearchByLongTermQualityTitle();
void renderSearchByComfortTitle();
void renderSearchResultTitle();
void renderNotFoundMsg();
void renderSortedMsg();
void renderSavedMsg();
void renderCarsListHeader();
void renderCarsListRow(const unsigned int number, const vehicle car);
void renderSearchResult(const unsigned int number, const vehicle car);
void renderSearchResultFooter(const unsigned int number);
void returnToMainMenu();
void displayMainMenu(const unsigned int listSize, const unsigned int catalogSize);

#endif

```

### 7. 4. 2. Листинг исходного кода файла реализации *view.c*

```

/**
 * Программное средство каталога автомобилей.
 * Разработчик - Дубинин А. В.

```

```

* 08.03.2017
*/

// Подключение заголовочных файлов
#include <stdio.h>
#include "../application/definitions.h"
#include "../application/types.h"
#include "view.h"

/**
 * Функция вывода на всю ширину окна горизонтальной полосы.
 */
void hr()
{
    int i;
    for (i = 0; i <= WINDOW_WIDTH; i++) {
        putchar(HYPHEN);
    }
    BR;
    return;
}

/**
 * Функция преобразования строки к верхнему регистру.
 * str - параметр типа string
 */
void toUpCase(char *str)
{
    char *buffer;
    for (buffer = str; *buffer != NULL_TERMINATED; buffer++) {
        *buffer = (char) toupper(*buffer);
    }
    return;
}

/**
 * Функция вывода на экран строки.
 * str - параметр-константа типа string
 */
void printString(const char *str)
{
    printf(PATTERN_STR, str);
    return;
}

/**
 * Функция вывода на экран заголовка.
 * title - параметр-константа типа string
 */
void renderTitle(const char *title)
{
    system(CALL_CLEAR);
    BR;
    printf(PATTERN_TITLE, title);
    hr();
    return;
}

/**
 * Функции вывода на экран строк в качестве заголовков.
 */

```

```

void renderListTitle()
{
    renderTitle(MSG_CATALOG_TITLE);
    return;
}

void renderEmptyListTitle()
{
    renderTitle(MSG_EMPTY_CATALOG_TITLE);
    return;
}

void renderAddCarTitle()
{
    renderTitle(MSG_ADD_CAR_TITLE);
    return;
}

void renderRemoveCarTitle()
{
    renderTitle(MSG_REMOVE_CAR_TITLE);
    return;
}

void renderSearchByWishesTitle()
{
    renderTitle(MSG_BY_WISHES_TITLE);
    return;
}

void renderSearchByFieldTitle()
{
    renderTitle(MSG_BY_FIELD_TITLE);
    return;
}

void renderSearchByModelTitle()
{
    renderTitle(MSG_BY_MODEL_TITLE);
    return;
}

void renderSearchByPriceTitle()
{
    renderTitle(MSG_BY_PRICE_TITLE);
    return;
}

void renderSearchByFuelConsumptionTitle()
{
    renderTitle(MSG_BY_FUEL_TITLE);
    return;
}

void renderSearchByLongTermQualityTitle()
{
    renderTitle(MSG_BY_QUALITY_TITLE);
    return;
}

void renderSearchByComfortTitle()
{

```

```

        renderTitle(MSG_BY_COMFORT_TITLE);
        return;
    }

void renderSearchResultTitle()
{
    renderTitle(MSG_FOUND_TITLE);
    return;
}

void renderNotFoundMsg()
{
    renderTitle(MSG_NOT_FOUND);
    return;
}

void renderSortedMsg()
{
    renderTitle(MSG_SORTED);
    return;
}

void renderSavedMsg()
{
    renderTitle(MSG_SAVED);
    return;
}

/**
 * Функция вывода на экран головной строки списка автомобилей.
 */
void renderCarsListHeader()
{
    printf(
        PATTERN_LIST_HEADER,
        COLUMN_NUMBER,
        COLUMN_MODEL,
        COLUMN_PRICE,
        COLUMN_FUEL,
        COLUMN_QUALITY,
        COLUMN_COMFORT
    );
    hr();
    return;
}

/**
 * Функция вывода на экран одной строки списка автомобилей.
 * number - параметр-константа типа integer (номер строки)
 * car - параметр-константа типа vehicle (структура с информацией об автомобиле)
 */
void renderCarsListRow(const unsigned int number, const vehicle car)
{
    printf(
        PATTERN_LIST_ROW,
        number,
        car.model,
        car.price,
        car.fuelConsumption,
        car.longTermQuality,
        car.comfort
    );
}

```

```

        return;
    }

/**
 * Функция вывода на экран одной строки списка найденных автомобилей.
 * number - параметр-константа типа integer (номер строки)
 * car - параметр-константа типа vehicle (структура с информацией об автомобиле)
 */
void renderSearchResult(const unsigned int number, const vehicle car)
{
    if (number == 1) {
        renderSearchResultTitle();
        renderCarsListHeader();
    }
    renderCarsListRow(number, car);
    return;
}

/**
 * Функция вывода на экран выбора действия после поиска автомобилей.
 * number - параметр-константа типа integer (количество найденных автомобилей)
 */
void renderSearchResultFooter(const unsigned int number)
{
    if (number == 0) {
        renderNotFoundMsg();
    } else {
        hr();
    }
    BR;
    puts(MSG_SEARCH_REPEAT);
    puts(MSG_RETURN_TO_MENU);
    return;
}

/**
 * Функция, реализующая возврат в главное меню программы.
 */
void returnToMainMenu()
{
    BR;
    puts(MSG_RETURN_TO_MENU);
    while (getchar() != ITEM_0) {}
    return;
}

/**
 * Функция отображения главного меню программы.
 * listSize - параметр-константа типа integer
 * (количество элементов динамического списка)
 * catalogSize - параметр-константа типа integer
 * (количество элементов типизированного файла каталога)
 */
void displayMainMenu(const unsigned int listSize, const unsigned int catalogSize)
{
    system(CALL_CLEAR);
    BR;

    // Для пустого списка можно только добавить новый элемент
    if (listSize > 0) {
        renderListTitle();
    }
}

```

```

        puts(MSG_SELECT_ACTION);
        BR;
        puts(MSG_DISPLAY_CATALOG);
        puts(MSG_ADD_CAR);
        puts(MSG_REMOVE_CAR);
        puts(MSG_SEARCH_BY_WISHES);
        puts(MSG_SEARCH_BY_FIELD);
        puts(MSG_SORT_BY_ALPHABET);
    } else {
        renderEmptyListTitle();
        puts(MSG_SELECT_ACTION);
        BR;
        puts(MSG_ADD_CAR);
    }
    BR;
    if (listSize > 0 || catalogSize > 0) {
        puts(MSG_SAVE_CATALOG);
    }
    puts(MSG_APP_EXIT);
    BR;
    hr();
    printString(MSG_DO_SELECT);
    return;
}

```