**­­МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В. И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

**Курсовая работа**

**по дисциплине «Программирование»**

**Тема: Разработка электронной картотеки.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2305 |  | Решетняк А.К. |
| Преподаватель |  | Хахаев И.А. |

Санкт-Петербург

2023

**Содержание**

[**Введение.** 3](#_Toc137124007)

[**Задание.** 3](#_Toc137124008)

[**Описание общей архитектуры данных.** 4](#_Toc137124009)

[**Связи между функциями** 5](#_Toc137124010)

[**Описание структур.** 6](#_Toc137124011)

[**Описание функций** 8](#_Toc137124012)

[**Описание переменных (для каждой функции).** 11](#_Toc137124013)

[**Схема алгоритма.** 29](#_Toc137124014)

[**Текст программы.** 50](#_Toc137124015)

[**Пример выполнения программы.** 77](#_Toc137124016)

[**Заключение.** 80](#_Toc137124017)

**Введение.**

Цель работы заключается в изучении языка программирования C и получение практических навыков программирования для создания электронной картотеки.

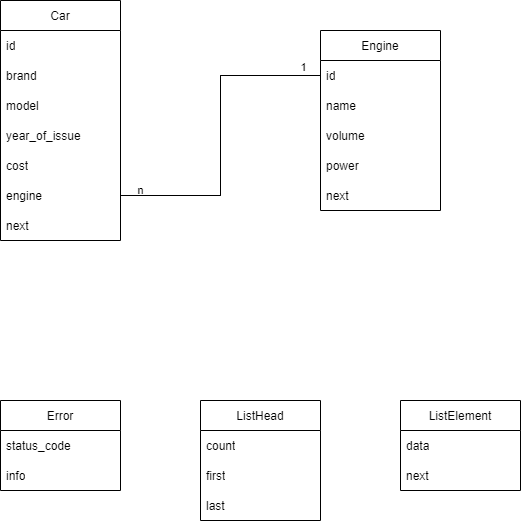
Задача работы представляет собой создание электронной картотеки автомобилей, содержащей таблицу с данными об автомобилях и таблицу с данными о двигателях

**Задание.**

Программа должна выполнять следующие действия:

* Выводить справку (вывод на экран краткой документации по пунктам меню).
* Добавлять новые карточки (внесения нового элемента).
* Редактирование карточки (изменение элемента, находящегося в списке)
* Удаление карточки (удаление элемента).
* Вывод картотеки (вывод на экран списка).
* Поиск карточки по параметру (поиск по введённым данным в заданном пользователем поле).
* Сортировка картотеки по параметру (сортировка по заданному пользователем полю)
* Выход (сохранение изменённой картотеки и завершение работы программы).

**Описание общей архитектуры данных.**



Структура Car – содержит данные об автомобилях, а также указатель на список двигателей

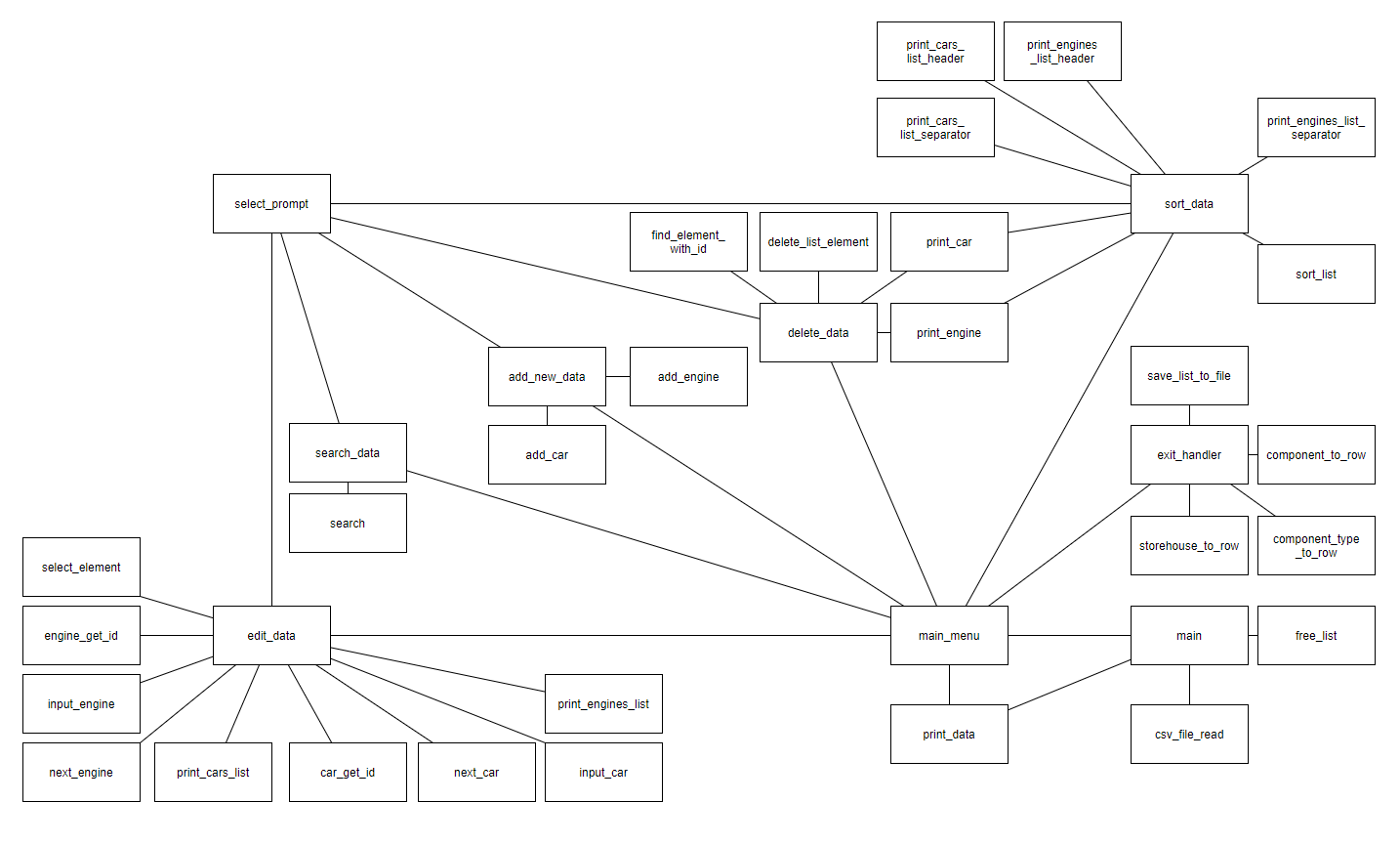
Структура Engine – содержит данные о двигателях

Структура Error – содержит код ошибки и информацию об ошибке

Структура ListHead – содержит количество элементов, а также указатели на первый и последний элементы

Структура ListElement – содержит данные об элементе и указатель на следующий элемент

**Связи между функциями**



**Описание структур.**

Структура error

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | status\_code | StatusCode | Код ошибки |
| 2 | info | char\* | Информация об ошибке |

Структура Engine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | id | unsigned | Номер id |
| 2 | name | char\* | Название двигателя |
| 3 | volume | float | Объем двигателя |
| 4 | power | int | Мощность двигателя |
| 5 | next | struct Engine\* | Указатель на следующий элемент |

Структура Car

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | id | unsigned | Номер id |
| 2 | brand | char\* | Марка |
| 3 | model | char\* | Модель |
| 4 | year\_of\_issue | int | Год производства |
| 5 | cost | float | Цена |
| 6 | engine | Engine\* | Двигатель |
| 7 | next | struct Car\* | Указатель на следующий элемент |

Структура ListHead

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | count | size\_t | Количество внесённых элементов |
| 2 | first | void\* | Указатель на первый элемент в списке |
| 3 | last | void\* | Указатель на последний элемент в списке |

Структура ListElement

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | data | void\* | Данные |
| 2 | next | void\* | Указатель на следующий элемент в списке |

**Описание функций**

**-** Error main\_menu(ListHead \*cars, ListHead \*engines)– главное меню

utils.h:

-FILE \*open\_or\_create\_file(char \*filename) – открытие файла или создание нового если файла нет

**-** Error csv\_file\_read(char \*filename, ListHead \*output, size\_t element\_size,

void \*\*(\*next\_getter)(void \*), void (\*free\_func)(void \*\*), Error (\*row\_converter)(CSVFileLine, void \*output, void \*), void \*additional\_args)– считывание файла и перевод информации в список

**-**void remove\_last\_symbol(char \*str) – удаление последнего символа строки

**-**void flush\_stdin() – удаление оставшихся символов переноса строки

structs\_to\_row.h:

**-** Error car\_to\_row(void \*car, CSVFileLine \*row) – перевод автомобиля в строку файла

**-** Error engine\_to\_row(void \*engine, CSVFileLine \*row) – перевод двигателя в строку файла

row\_to\_struct.h:

**-** Error row\_to\_engine(CSVFileLine row, void \*engine, ...) – перевод строки файла в структуру двигателя

**-** Error row\_to\_car(CSVFileLine row, void \*car, void \*additional\_args) – перевод строки файла в структуру автомобиля

printers.h:

- void print\_car(const void \*car) – вывод автомобиля

- void print\_cars\_list\_separator() – вывод разделителя таблицы

- void print\_cars\_list\_header() – вывод заголовка таблицы

- void print\_cars\_list(const ListHead cars) – вывод списка автомобилей

- void print\_engine(const void \*engine) – вывод двигателя

-void print\_engines\_list\_separator() – вывод разделителя таблицы

-void print\_engines\_list\_header() – вывод заголовка таблицы

-void print\_engines\_list(const ListHead engines) – вывод списка двигателей

menu\_functions.h:

-void about() – вывод меню помощи

- Error add\_new\_data(ListHead \*cars, ListHead \*engines) – добавление данных в картотеку

- Error edit\_data(ListHead cars, ListHead engines) – редактирует данные в картотеке

- Error delete\_data(ListHead \*cars, ListHead \*engines)– удаление данных из картотеки

- void print\_data(const ListHead cars, const ListHead engines) – вывод картотеки

- Error search\_data(ListHead cars, ListHead engines) – поиск данных в картотеке

- Error sort\_data(ListHead cars, ListHead engines) – сортировка данных картотеки

- Error exit\_handler(ListHead cars, ListHead engines) – выход из программы и сохранение данных в картотеку по желанию пользователя

-int select\_prompt(char \*prompt, int min\_value, int max\_value) – выбор пункта в меню

getters.h:

- unsigned \*car\_get\_engine\_id(void \*car) – получение id двигателя автомобиля

- char \*\*car\_get\_engine\_name(void \*car) – получение названия двигателя автомобиля

- float \*car\_get\_cost(void \*car) – получение цены автомобиля

- char \*\*car\_get\_model(void \*car) – получение модели автомобиля

- unsigned \*car\_get\_id(void \*car) – получение id автомобиля

- char \*\*car\_get\_brand(void \*car) – получение марки автомобиля

- int \*car\_get\_year\_of\_issue(void \*car) – получение года выпуска автомобиля

- void \*\*next\_car(void \*car) – переход к следующему автомобилю

- unsigned \*engine\_get\_id(void \*engine) – получение id двигателя

- char \*\*engine\_get\_name(void \*engine) – получение названия двигателя

- float \*engine\_get\_volume(void \*engine) – получение объема двигателя

- int \*engine\_get\_power(void \*engine) – получение мощности двигателя

- void \*\*next\_engine(void \*engine) – переход к следующему двигателю

- void \*\*list\_element\_get\_data(void \*list\_element) – получение данных элемента

- void \*\*next\_list\_element(void \*list\_element) – переход к следующему элементу

csv\_file\_to\_list.h:

-Error csv\_file\_to\_list(CSVFile csv, ListHead \*output, size\_t element\_size, void \*\*(\*next\_getter)(void \*), void (\*free\_func)(void \*\*), Error (\*row\_converter)(CSVFileLine, void \*output, void \*), void \*additional\_args) – перевод файла в структуру

csv\_file.h:

-void CSVFileFree(CSVFile \*file) – освобождение памяти файла

-int CSVFileRead(FILE \*fp, CSVFile \*output) – чтение файла

-void CSVFileWrite(FILE \*fp, CSVFile data) – запись в файл

-void \_CSVFileFreeLine(CSVFileLine \*line, size\_t fields\_count) – освобождение памяти строки файла

-int \_CSVFileAddCharToOutput(char character, char \*\*output, size\_t \*string\_len, size\_t \*allocated) – добавление символа в конец строки

-int \_CSVFileReadField(FILE \*csv\_file, char \*\*output) – чтение поля из файла

-int \_CSVFileCountFields(FILE \*csv\_file, size\_t \*fields\_count) – подсчет количества полей в файле

-int \_CSVFileSkipCRLF(FILE \*csv\_file) – пропуск окончания строки

-int \_CSVReadLine(FILE \*csv\_file, CSVFileLine \*output, size\_t fields\_count) – чтение строки файла

add\_funcs.h:

- Error add\_car(ListHead \*cars, const ListHead engines) – добавление автомобиля

- Error add\_engine(ListHead \*engines) – добавление двигателя

edit\_funcs.h:

-void \*select\_element(const ListHead list, unsigned \*(\*id\_getter)(void \*), void \*\*(next\_getter)(void \*), void (\*printer)(const void \*)) – выбор элемента

exit\_funcs.h:

-Error save\_list\_to\_file(ListHead list, char \*output\_filename, size\_t element\_fields\_count, Error (\*element\_to\_row)(void \*element, CSVFileLine \*row), void \*\*(\*next\_getter)(void \*element)) – сохранение картотеки в файл

input\_funcs.h:

- Car \*input\_car(const ListHead engines) – ввод автомобиля

- Engine \*input\_engine() – ввод двигателя

search\_funcs.h:

-char search(const ListHead list, void \*field\_value, size\_t field\_size, void \*(\*field\_getter)(void \*), void \*\*(\*next\_getter)(void \*), void (\*printer)(const void \*), char is\_str) – поиск

sort\_funcs.h:

-Error sort\_list(ListHead to\_sort, ListHead \*output, void \*(\*field\_getter)(void \*), void \*\*(\*next\_getter)(void \*), VariableType type) – сортировка списка

-int compare(void \*value1, void \*value2, VariableType type) – сравнение элементов

utils.h:

-void \*find\_element\_with\_id(ListHead list, unsigned id, unsigned \*(\*id\_getter)(void \*), void \*\*(\*next\_getter)(void \*)) – поиск элемента с заданным id

-void delete\_list\_element(ListHead \*list, void \*element, void \*\*(\*next\_getter)(void \*), void (\*free\_func)(void \*\*)) – удаление элемента

csv\_row\_converters.h:

-Error scan\_row(CSVFileLine row, const char \*format, ...) – считывание строки файла

-Error create\_row(CSVFileLine \*row, const char \*format, ...) – запись строки в файл

utils.h:

-Error format\_str\_to\_enum\_list(const char \*format, int \*\*output, size\_t \*count) – конвертация строки спецификаторов в массив, состоящий из элементов перечисления этих спецификаторов

-size\_t specifier\_to\_size(int int\_specifier) – конвертация элемента перечисления спецификаторов в размер требуемой памяти для хранения спецификатора такого типа

-int specifier\_to\_enum(char specifier[3]) – конвертация одного спецификатора в один элемент перечисления спефикаторов

**Описание переменных (для каждой функции).**

main

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | error | Error | Код ошибки |
| 2 | engines | ListHead | Список двигателей |
| 3 | cars | ListHead | Список автомобилей |

main\_menu

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead\* | Указатель на список автомобилей |
| 2 | engines | ListHead\* | Указатель на список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | menu\_selection | MenuSelection | Выбор пункта меню |

add\_new\_data

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead\* | Указатель на список автомобилей |
| 2 | engines | ListHead\* | Указатель на список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | table\_selection | int | Выбор таблицы |

edit\_data

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead\* | Указатель на список автомобилей |
| 2 | engines | ListHead\* | Указатель на список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | table\_selection | int | Выбор таблицы |
| 5 | element | void\* | Элемент |
| 6 | replacement | void\* | Замена элемента |

delete\_data

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead\* | Указатель на список автомобилей |
| 2 | engines | ListHead\* | Указатель на список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | id | unsigned | Номер id |
| 5 | iter | void\* | Вспомогательный элемент |
| 6 | found | char | Флаг нахождения |
| 7 | table\_selection | int | Выбор таблицы |

print\_data

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead | Список компонентов |
| 2 | engines | ListHead | Список типов компонентов |

search\_data

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead\* | Указатель на список автомобилей |
| 2 | engines | ListHead\* | Указатель на список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | car\_field\_getters[] | void\* | Получение данных с полей структуры |
| 5 | engine\_field\_getters[] | void\* | Получение данных с полей структуры |
| 7 | search\_criteria | int | Критерий поиска |
| 8 | is\_str | char | Флаг является ли элемент строкой |
| 9 | found | char | Флаг нахождения |
| 10 | value\_size | size\_t | Размер типа |
| 11 | buf[1024] | char | Буфер |
| 12 | table\_selection | int | Выбор таблицы |

sort\_data

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead\* | Указатель на список автомобилей |
| 2 | engines | ListHead\* | Указатель на список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | table\_selection | int | Выбор таблицы |
| 5 | sort\_criteria | int | Критерий поиска |
| 6 | sort\_type | VariableType | Тип данных для поиска |
| 7 | sorted | ListHead | Отсортированный список |
| 8 | element | ListElement\* | Элемент |
| 9 | car\_field\_getters[] | void\* | Получение данных с полей структуры |
| 10 | engine\_field\_getters[] | void\* | Получение данных с полей структуры |
| 11 | printer() | void\* | Вывод структуры |
| 12 | separator\_printer() | void\* | Вывод разделителей таблицы |
| 13 | header\_printer() | void\* | Вывод заголовка таблицы |

exit\_handler

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead | Список автомобилей |
| 2 | engines | ListHead | Список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | confirm | char | Подтверждение пользователем |

select\_prompt

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | prompt | char\* | Ответ пользователя |
| 2 | min\_value | int | Минимальное значение |
| 3 | max\_value | int | Максимальное значение |
| 4 | table\_selection | int | Выбор таблицы |

print\_car

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |
| 2 | car\_struct | Car\* | Переменная для вывода элементов списка |

print\_cars\_list

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead | Список автомобилей |
| 2 | car | Car\* | Переменная для вывода списка |

print\_engine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\* | Указатель на список двигателей |
| 2 | engine\_struct | Engine\* | Переменная для вывода элементов списка |

print\_engines\_list

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engines | ListHead | Список двигателей |
| 2 | engine | Engine\* | Переменная для вывода списка |

add\_car

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | cars | ListHead\* | Указатель на список автомобилей |
| 2 | engines | ListHead\* | Указатель на список двигателей |
| 3 | error | Error | Код ошибки |
| 4 | new\_car | Car\* | Новый автомобиль |

add\_engine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engines | ListHead\* | Указатель на список двигателей |
| 2 | error | Error | Код ошибки |
| 3 | new\_engine | ComponentType\* | Новый двигатель |

select\_element

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list | ListHead | Список элементов |
| 2 | (\*id\_getter)(void\*) | unsigned\* | Получение id |
| 3 | (next\_getter)(void\*) | void\*\* | Получение следующего элемента |
| 4 | (\*printer)(const void\*) | void | Вывод элемента |
| 5 | confirm | char | Подтверждение пользователем |
| 6 | element | void\* | Элемент |
| 7 | id | unsigned | Номер id |

save\_list\_to\_file

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list | ListHead | Список элементов |
| 2 | output\_filename | char\* | Имя выходного файла |
| 3 | element\_fields\_count | size\_t | Количество полей элементов |
| 4 | (\*element\_to\_row)(void \*element, CSVFileLine \*row) | Error | Преобразование элемента в строку файла |
| 5 | error | Error | Код ошибки |
| 6 | (\*next\_getter)(void \*element) | void\*\* | Получение следующего элемента |
| 7 | element | void\* | Элемент |
| 8 | csv | CSVFile | Файл |
| 9 | output | FILE\* | Выходной файл |
| 10 | i | size\_t | Переменная для цикла for |

input\_car

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | Car\* | Указатель на автомобиль |
| 2 | engines | ListHead | Список двигателей |

input\_engine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | Engine\* | Указатель на список двигателей |

search

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list | ListHead | Список элементов |
| 2 | field\_value | void\* | Значение поля |
| 3 | field\_size | size\_t | Размер поля |
| 4 | (\*printer)(const void\*) | void | Вывод элемента |
| 5 | (\*next\_getter)(void \*) | void\*\* | Получение следующего элемента |
| 6 | (\*field\_getter)(void\*) | void\* | Получение поля |
| 7 | is\_str | char | Флаг является ли строкой |
| 8 | iter | void\* | Вспомогательная переменная |
| 9 | found | char | Флаг нахождения |

sort\_list

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | to\_sort | ListHead | Список для сортировки |
| 2 | output | ListHead\* | Указатель на выходной список |
| 3 | type | VariableType | Тип переменной |
| 4 | error | Error | Код ошибки |
| 5 | (\*next\_getter)(void \*) | void\*\* | Получение следующего элемента |
| 6 | (\*field\_getter)(void\*) | void\* | Получение поля |
| 7 | element | void\* | Элемент |
| 8 | element2 | void\* | Элемент 2 |
| 9 | temp | void\* | Временная переменная |

compare

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | value1 | void\* | Значение 1 |
| 2 | value2 | void\* | Значение 2 |
| 3 | type | VariableType | Тип переменной |
| 4 | result | int | Результат |

find\_element\_with\_id

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list | ListHead | Список элементов |
| 2 | id | unsigned | Номер id |
| 3 | (\*id\_getter)(void\*) | unsigned\* | Получение id |
| 4 | (\*next\_getter)(void \*) | void\*\* | Получение следующего элемента |
| 5 | element | void\* | Элемент |
| 6 | found | char | Флаг нахождения |

delete\_list\_element

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list | ListHead\* | Список элементов |
| 2 | element | void\* | Элемент |
| 3 | (\*free\_func)(void\*\*) | void | Освобождение функции |
| 4 | (\*next\_getter)(void \*) | void\*\* | Получение следующего элемента |
| 5 | iter | void\* | Вспомогательная переменная |
| 6 | prev | void\* | Предыдущий элемент |

car\_get\_engine\_id

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

car\_get\_engine\_name

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

car\_get\_cost

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

car\_get\_model

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

car\_get\_id

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

car\_get\_brand

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

car\_get\_year\_of\_issue

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

next\_car

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на список автомобилей |

engine\_get\_id

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\* | Указатель на список двигателей |

engine\_get\_name

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\* | Указатель на список двигателей |

engine\_get\_volume

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\* | Указатель на список двигателей |

engine\_get\_power

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\* | Указатель на список двигателей |

next\_engine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\* | Указатель на список двигателей |

list\_element\_get\_data

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list\_element | void\* | Указатель на список элементов |

next\_list\_element

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list\_element | void\* | Указатель на список элементов |

car\_to\_row

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\* | Указатель на автомобиль |
| 2 | row | CSVFileLine\* | Строка файла |
| 3 | car\_struct | Car\* | Указатель на список автомобилей |

engine\_to\_row

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\* | Указатель на двигатель |
| 2 | row | CSVFileLine\* | Строка файла |
| 3 | engine\_struct | Engine\* | Указатель на список двигателей |

flush\_stdin

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | c | char | Получение символа |

remove\_last\_symbol

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | str | char\* | Строка |
| 2 | len | size\_t | Длина строки |

csv\_file\_read

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | filename | char\* | Имя файла |
| 2 | output | ListHead\* | Выходной список |
| 3 | element\_size | size\_t | Размер элемента |
| 4 | (\*next\_getter)(void\*) | void\*\* | Получение следующего элемента |
| 5 | (\*free\_func)(void\*\*) | void | Освобождение функции |
| 6 | (\*row\_converter)(CSVFileLine,  void\*, va\_list) | Error | Перевод строк файла |
| 7 | f | FILE\* | Файл |
| 8 | error | Error | Код ошибки |
| 9 | csv | CSVFile | CSV-файл |
| 10 | csv\_status | int | Статус файла |
| 11 | additional\_args | va\_list | Дополнительные аргументы |

open\_or\_create\_file

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | filename | char\* | Имя файла |
| 2 | error | StatusCode | Код ошибки |
| 3 | f | FILE\* | Файл |

free\_car

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | car | void\*\* | Указатель на автомобиль |
| 2 | car\_struct | Car\* | Указатель на список автомобилей |

free\_engine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | engine | void\*\* | Указатель на двигатель |
| 2 | engine\_struct | Engine\* | Указатель на список двигателей |

free\_list\_element

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list\_element | void\*\* | Указатель на элемент списка |

free\_list

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | list | ListHead\* | Указатель на список элементов |
| 2 | (\*free\_func)(void\*\*) | void | Освобождение функции |
| 3 | (\*next\_func)(void\*) | void\*\* | Переход к следующей функции |

row\_to\_engine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | row | CSVFileLine | Строка файла |
| 2 | engine | void\* | Указатель на двигатель |
| 3 | pseudo | Engine\* | Указатель на вспомогательный двигатель |

row\_to\_car

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | row | CSVFileLine | Строка файла |
| 2 | car | void\* | Указатель на автомобиль |
| 3 | additional\_args | void\* | Дополнительные аргументы |
| 4 | engines | ListHead | Список двигателей |
| 5 | engine\_id | unsigned | Номер id двигателя |
| 6 | error | Error | Код ошибки |
| 7 | engine | Engine\* | Указатель на двигатель |
| 8 | pseudo | Car\* | Указатель на вспомогательный автомобиль |

csv\_file\_to\_list

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | csv | CSVFile | Файл |
| 2 | output | ListHead\* | Указатель на выходной список |
| 3 | element\_size | size\_t | Размер элемента |
| 4 | (\*next\_getter)(void \*) | void\*\* | Получение следующего элемента |
| 5 | (\*free\_func)(void \*\*) | void | Освобождение функции |
| 6 | (\*row\_converter)(CSVFileLine, void \*output, void \* | Error | Преобразование строк файла |
| 7 | additional\_args | void\* | Дополнительные аргументы |
| 8 | error | Error | Код ошибки |
| 9 | i | size\_t | Переменная для цикла for |

CSVFileFree

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | file | CSVFile\* | Файл |
| 2 | i | size\_t | Переменная для цикла for |

CSVFileRead

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | output | CSVFile\* | Указатель на выходной файл |
| 2 | fp | FILE\* | Файл |
| 3 | error | int | Код ошибки |
| 4 | continue\_loop | int | Флаг продолжения |
| 5 | allocated | size\_t | Переменная для выделения памяти |
| 6 | realloc\_result | CSVFileLine\* | Результат |
| 7 | next\_char | int | Следующий символ |

CSVFileWrite

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | data | CSVFile | Файл данных |
| 2 | fp | FILE\* | Файл |
| 3 | i | size\_t | Переменная для цикла for |
| 4 | j | size\_t | Переменная для цикла for |
| 5 | CRLF | char | Переменная для записи в файл |

\_CSVFileFreeLine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | line | CSVFileLine\* | Строка файла |
| 2 | fields\_count | size\_t | Количество полей |
| 3 | i | size\_t | Переменная для цикла for |

\_CSVFileAddCharToOutput

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | character | char | Символ |
| 2 | output | char\*\* | Выходная строка |
| 3 | string\_len | size\_t\* | Длина строки |
| 4 | allocated | size\_t\* | Переменная для выделения памяти |
| 5 | error | int | Код ошибки |
| 6 | realloc\_result | char\* | Результат |

\_CSVFileReadField

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | csv\_file | FILE\* | Файл |
| 2 | output | char\*\* | Выходная строка |
| 3 | string\_len | size\_t | Длина строки |
| 4 | allocated | size\_t | Переменная для выделения памяти |
| 5 | i | size\_t | Переменная для цикла for |
| 6 | quoted | char | Флаг, показывающий кавычки |
| 7 | continue\_loop | char | Флаг продолжения |
| 8 | realloc\_result | char\* | Результат |
| 9 | current\_char | int | Текущий символ |
| 10 | error | int | Код ошибки |

\_CSVFileCountFields

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | csv\_file | FILE\* | Файл |
| 2 | fields\_count | size\_t\* | Количество полей |
| 3 | start | long | Старт |
| 4 | current\_char | int | Текущий символ |
| 5 | previous\_char | int | Предыдущий символ |
| 6 | quoted | char | Флаг, показывающий кавычки |
| 7 | continue\_loop | char | Флаг продолжения |
| 8 | error | int | Код ошибки |
| 9 | quotes\_count | size\_t | Количество кавычек |

\_CSVFileSkipCRLF

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | csv\_file | FILE\* | Файл |
| 2 | error | int | Код ошибки |
| 3 | next | int | Переход к следующему символу |

\_CSVReadLine

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | csv\_file | FILE\* | Файл |
| 2 | output | CSVFileLine\* | Выходная строка |
| 3 | fields\_count | size\_t | Количество полей |
| 4 | error | int | Код ошибки |
| 5 | i | size\_t | Переменная для цикла for |
| 6 | next\_char | int | Следующий символ |
| 7 | continue\_loop | char | Флаг продолжения |

scan\_row

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | row | CSVFileLine | Строка файла |
| 2 | format | char\* | Формат |
| 3 | specifiers | COURSE\_TASK\_  SPECIFIER\_TYPE\* | Спецификаторы |
| 4 | count | size\_t | Количество |
| 5 | i | size\_t | Переменная для цикла for |
| 6 | error | Error | Код ошибки |
| 7 | args | va\_list | Аргументы |
| 8 | str | char\*\* | Строка |
| 9 | temp | int | Временная переменная |

create\_row

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | row | CSVFileLine\* | Строка файла |
| 2 | format | char\* | Формат |
| 3 | specifiers | COURSE\_TASK\_  SPECIFIER\_TYPE\* | Спецификаторы |
| 4 | count | size\_t | Количество |
| 5 | i | size\_t | Переменная для цикла for |
| 6 | error | Error | Код ошибки |
| 7 | length | size\_t | Длина |
| 8 | args | va\_list | Аргументы |
| 9 | str | char\* | Строка |
| 10 | temp | int | Временная переменная |

format\_str\_to\_enum\_list

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | output | int\*\* | Выходное значение |
| 2 | format | char\* | Формат |
| 3 | buf | char[3] | Буфер |
| 4 | count | size\_t\* | Количество |
| 5 | i | size\_t | Переменная для цикла for |
| 6 | error | Error | Код ошибки |
| 7 | allocated | size\_t | Переменная для выделения памяти |
| 8 | j | size\_t | Переменная для цикла for |
| 9 | temp\_specifiers\_ptr | int\* | Временные спецификаторы |
| 10 | temp | int | Временная переменная |

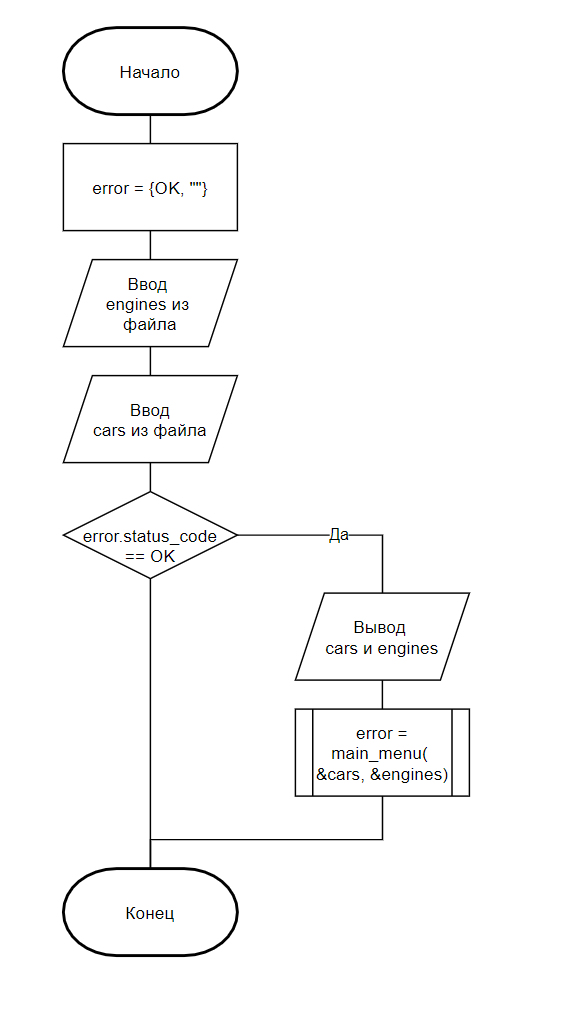
specifier\_to\_size

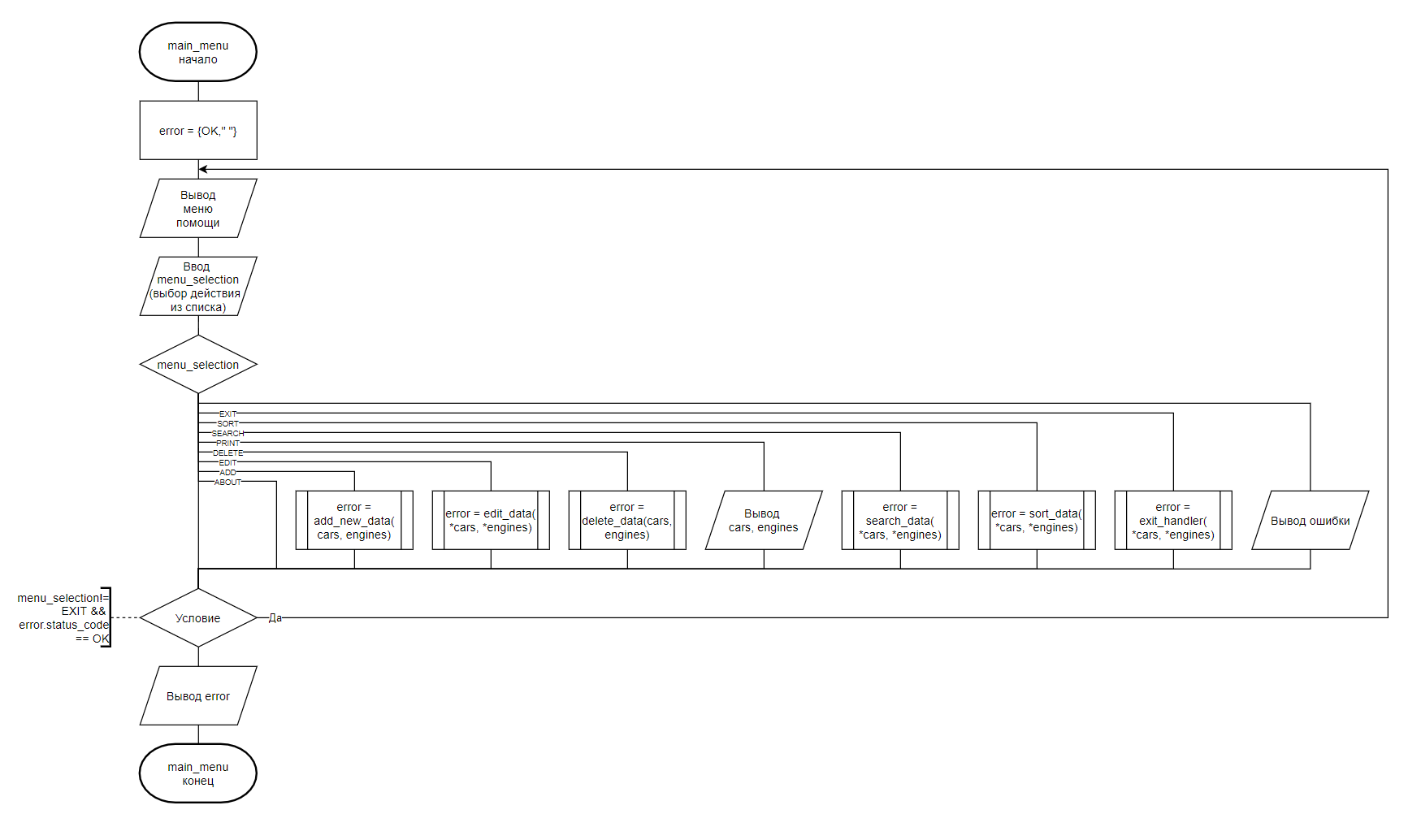
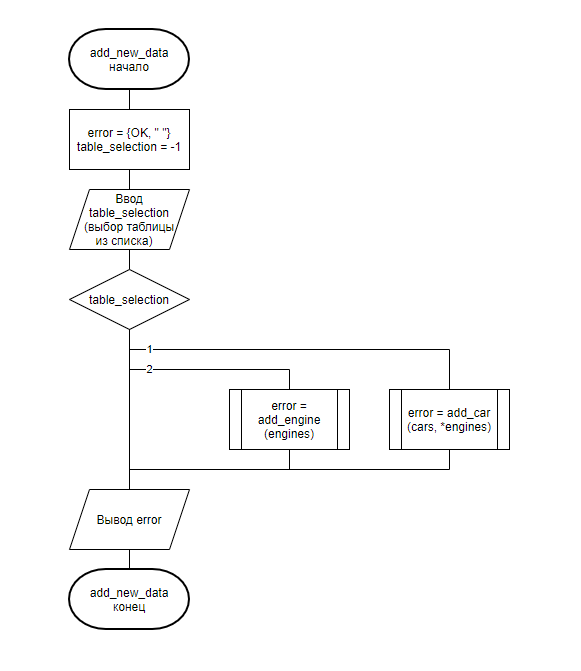
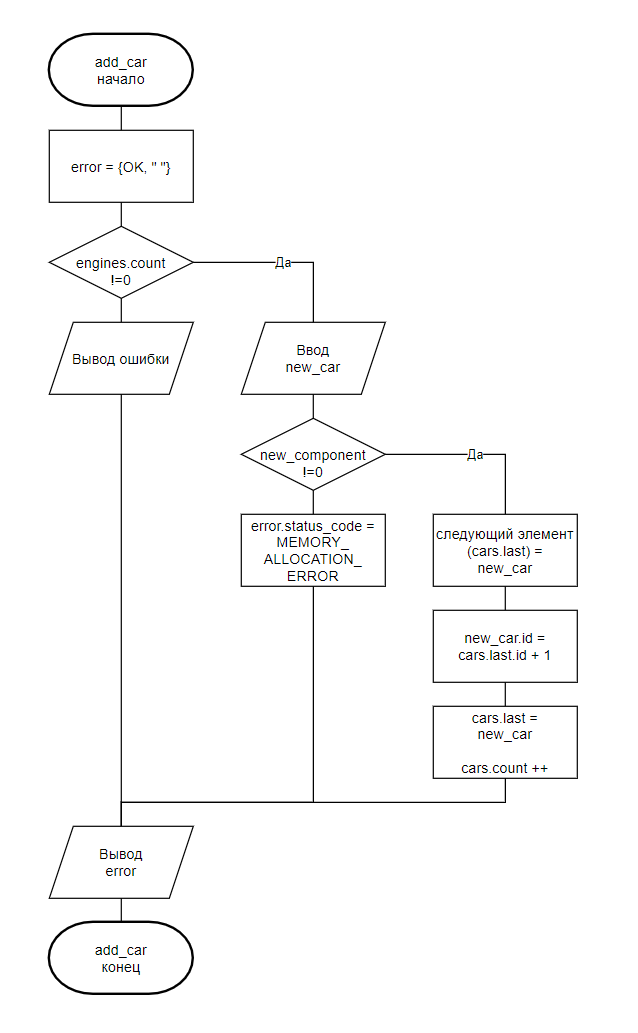
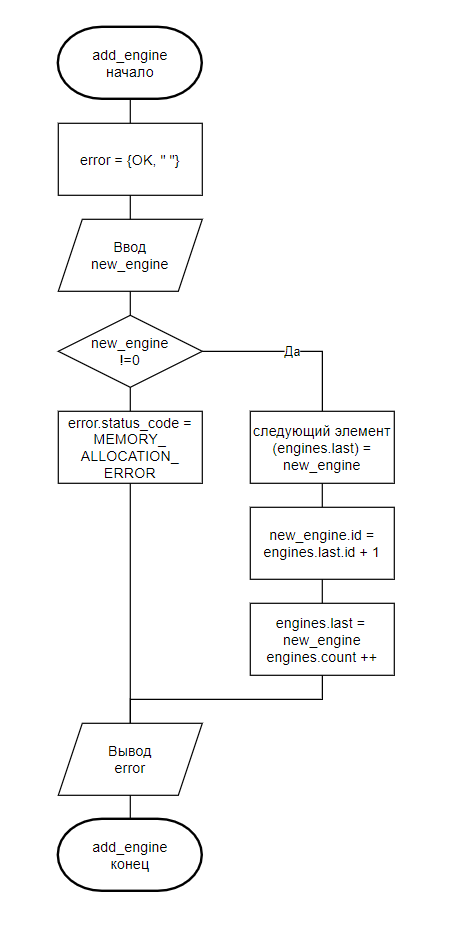
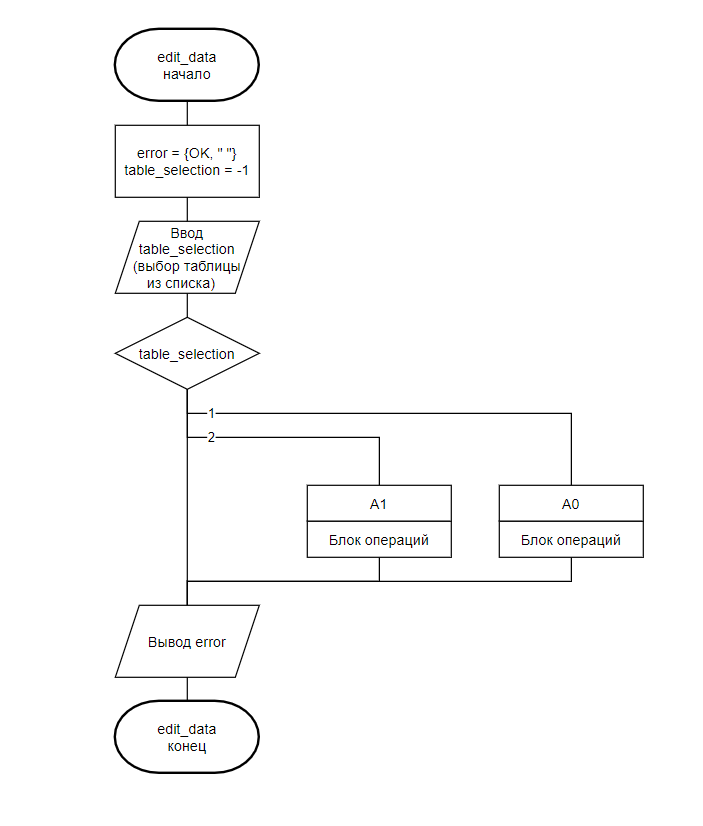
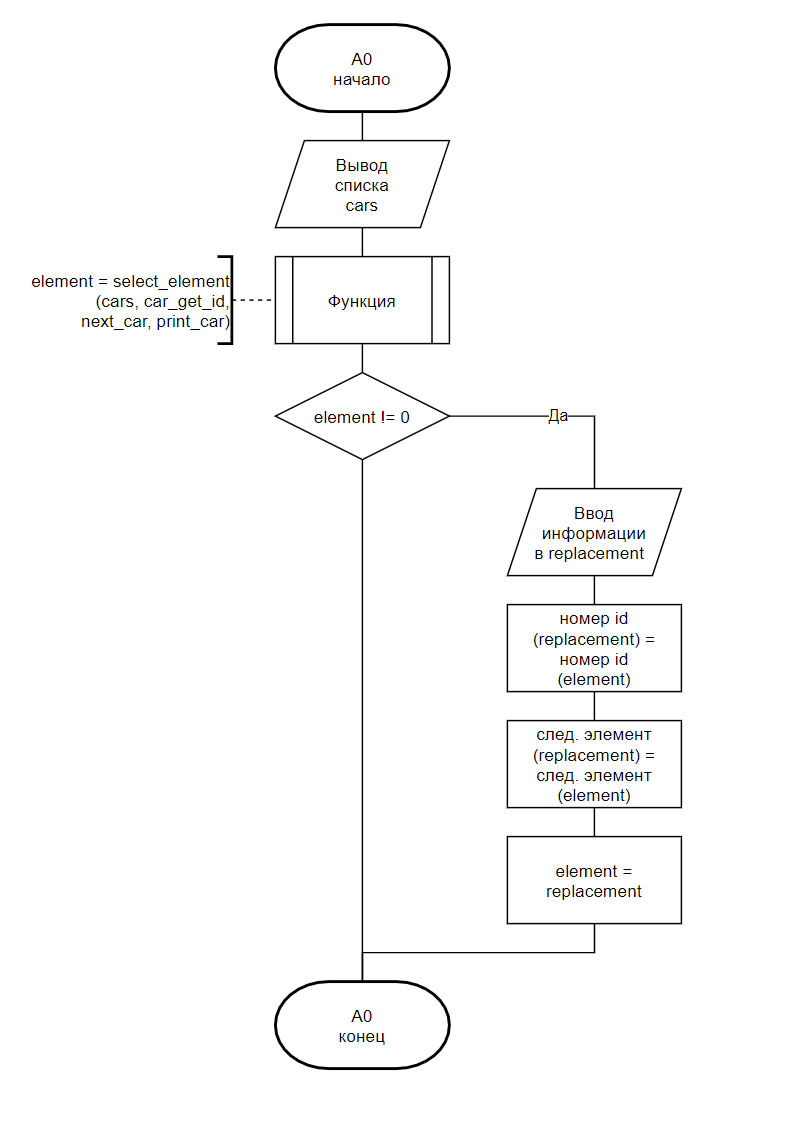
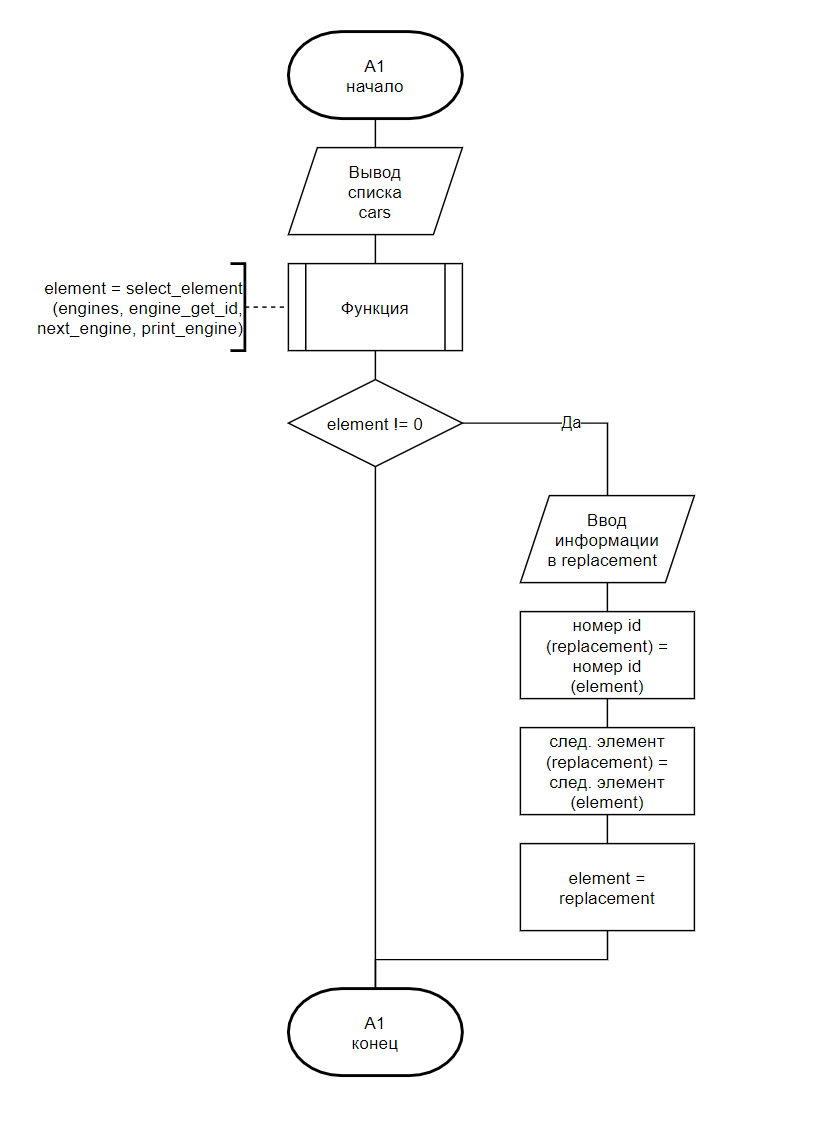
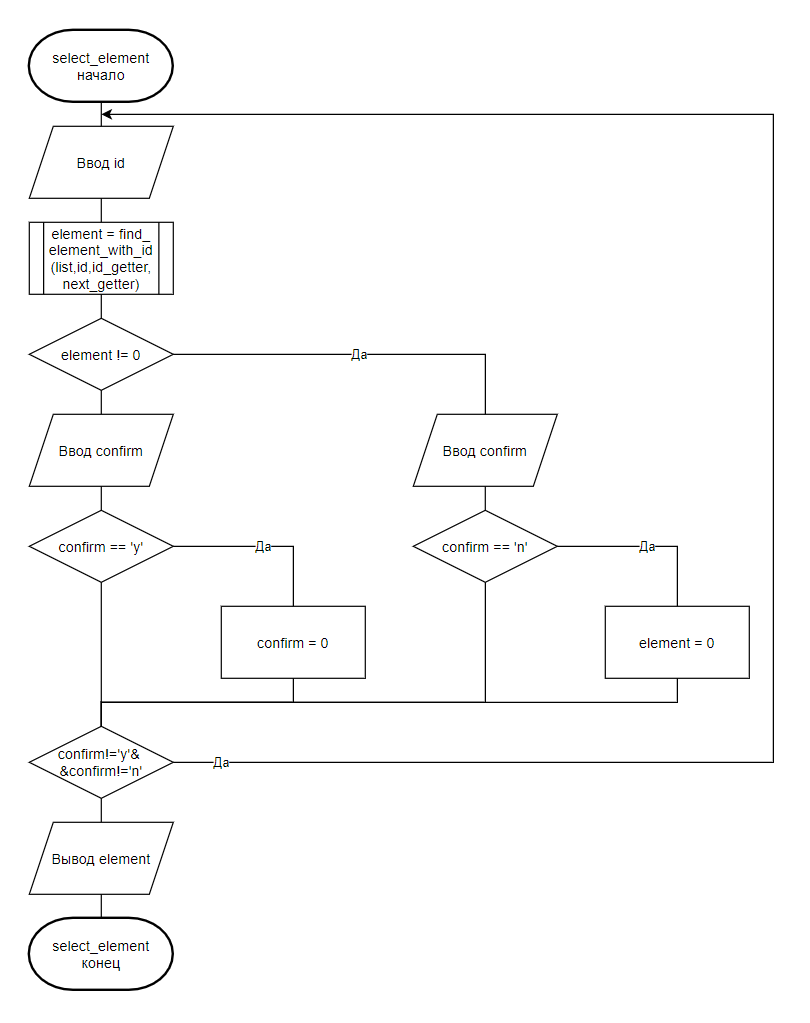
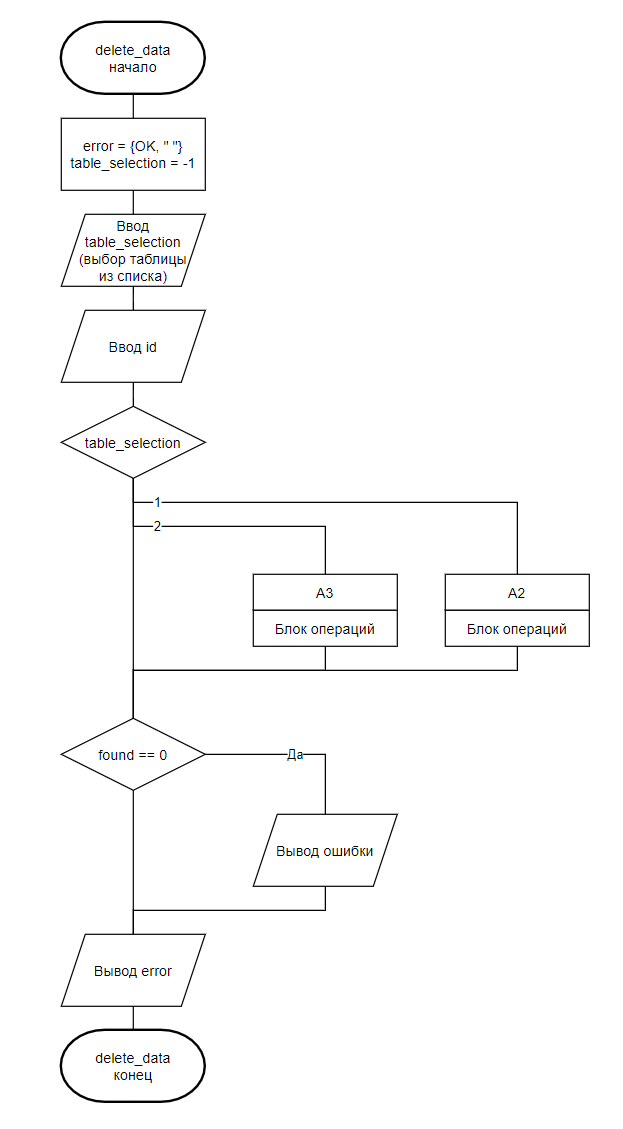
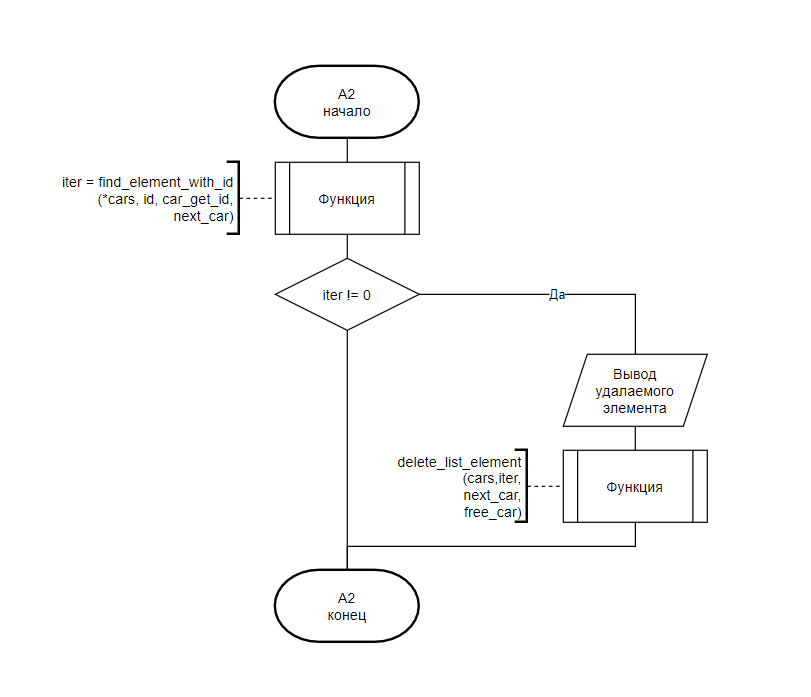
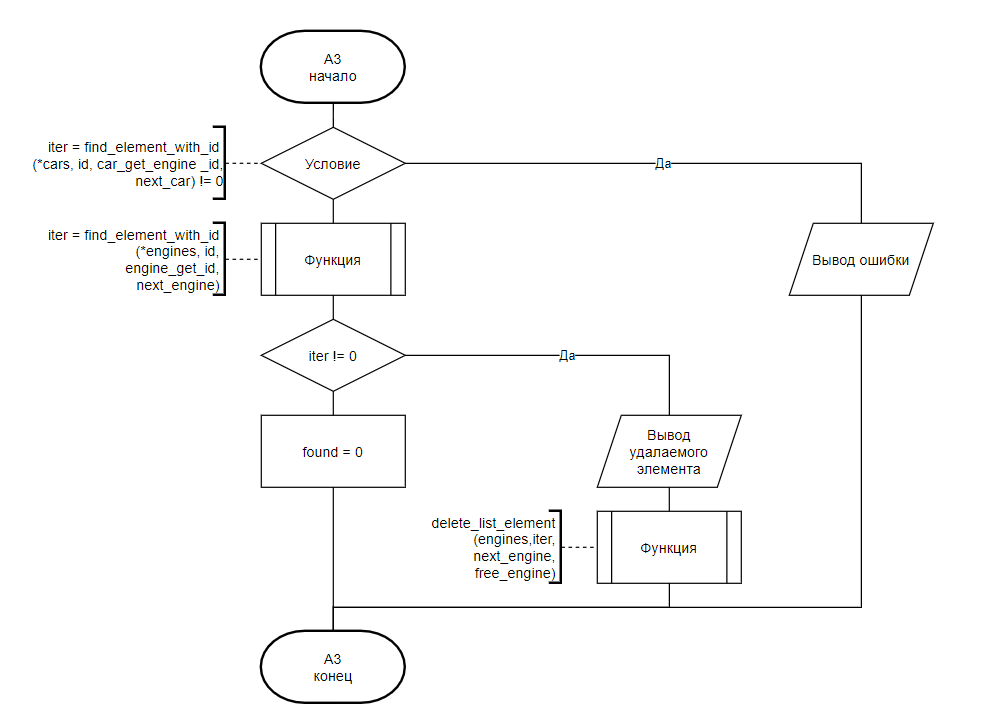
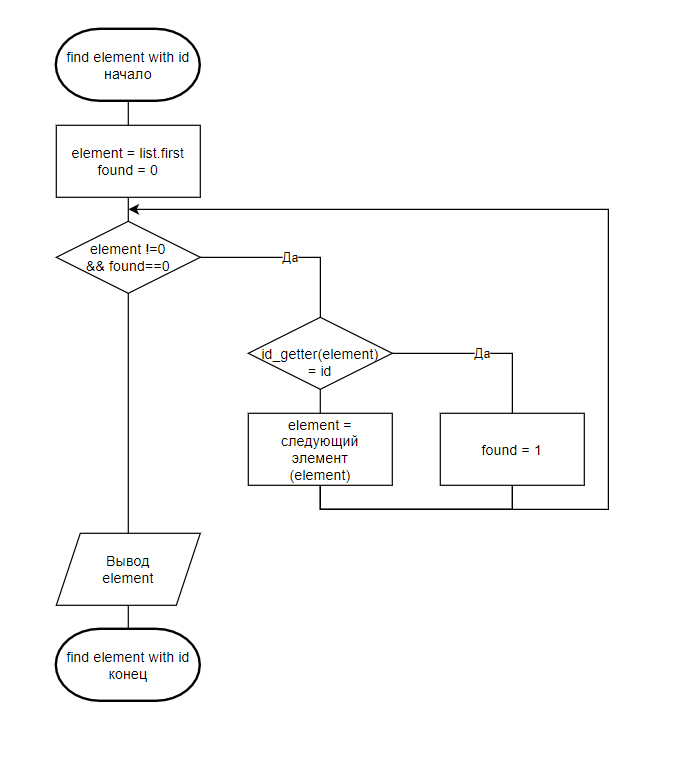
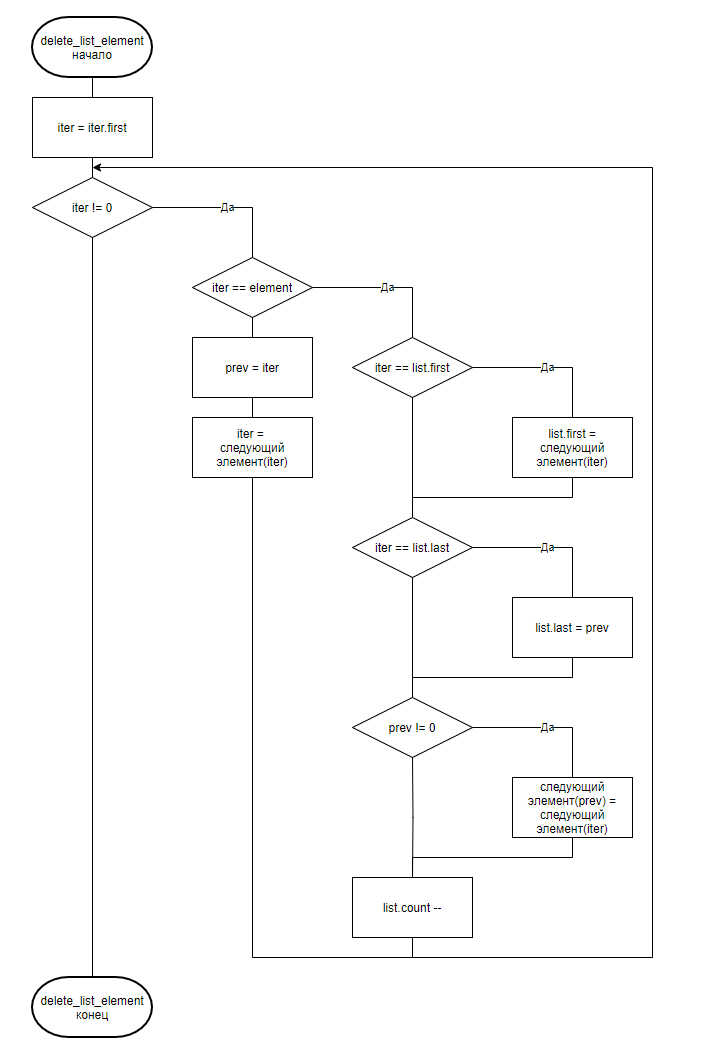
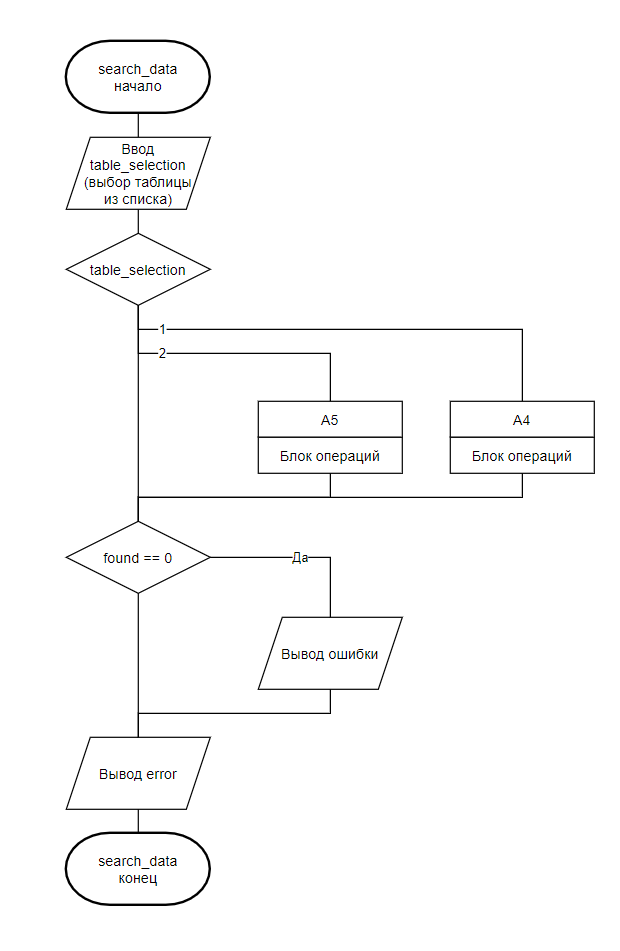
|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | int\_specifier | int | Целочисленный спецификатор |
| 2 | specifier | COURSE\_TASK\_ SPECIFIER\_TYPE | Спецификатор |
| 3 | size | size\_t | Размер |

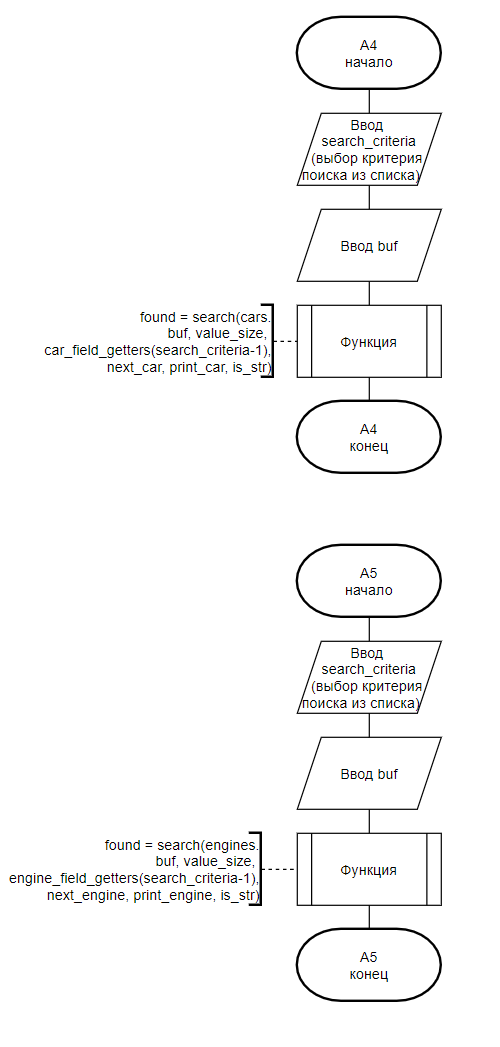
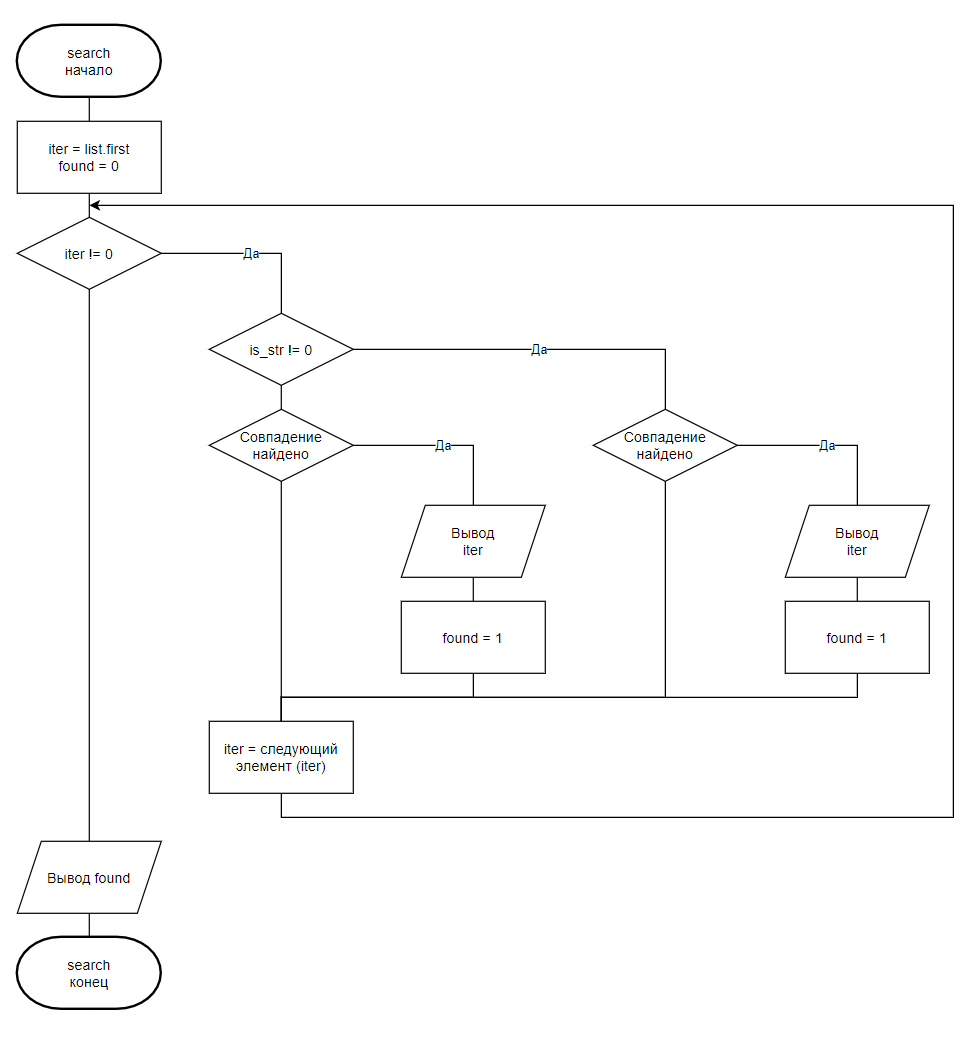
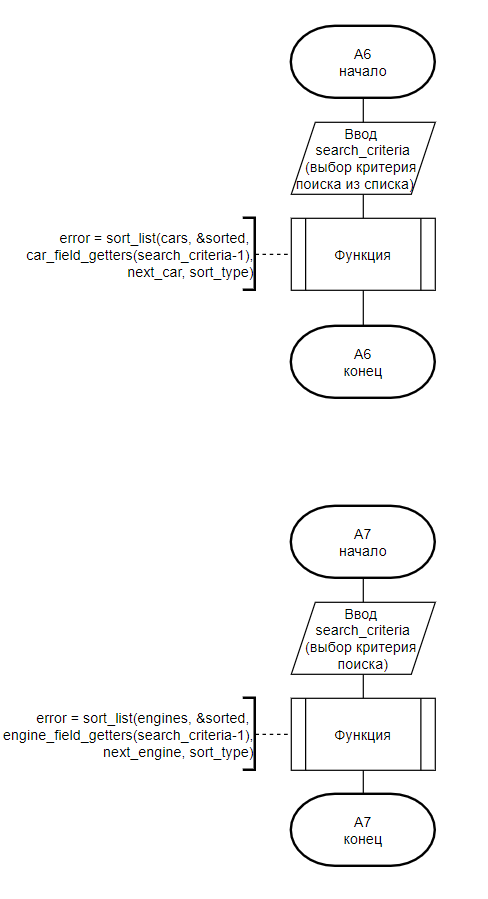
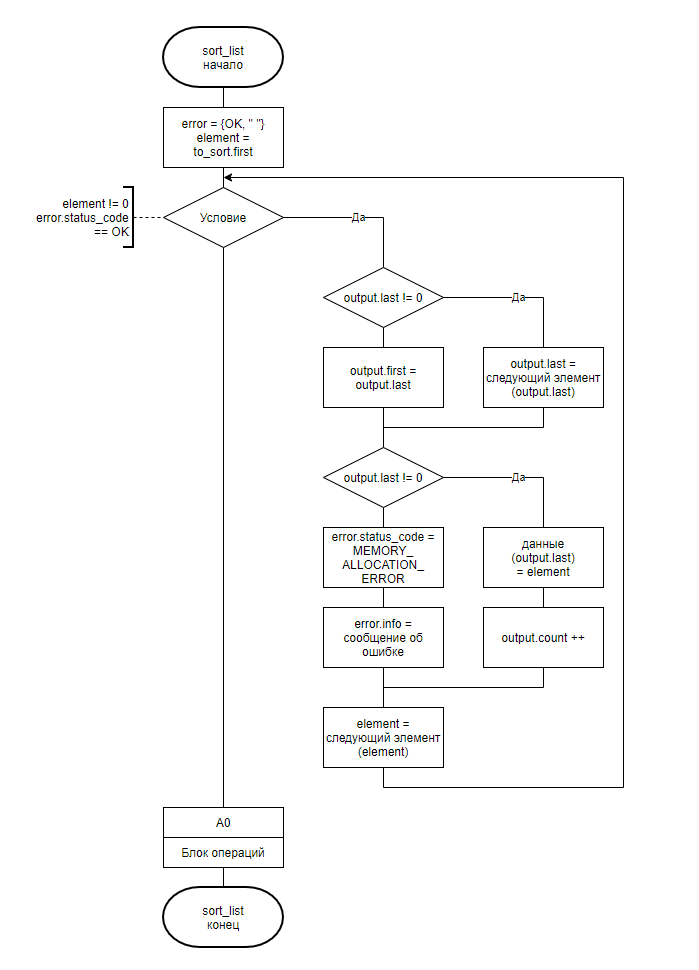
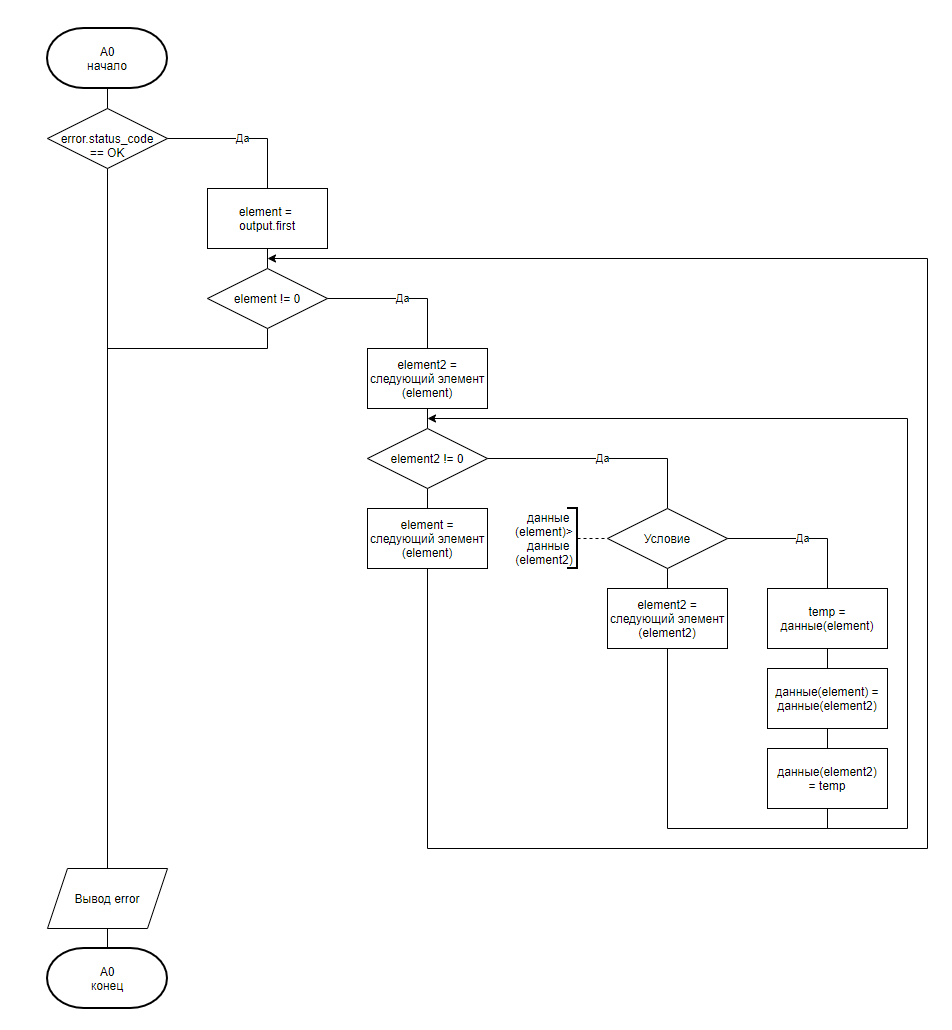
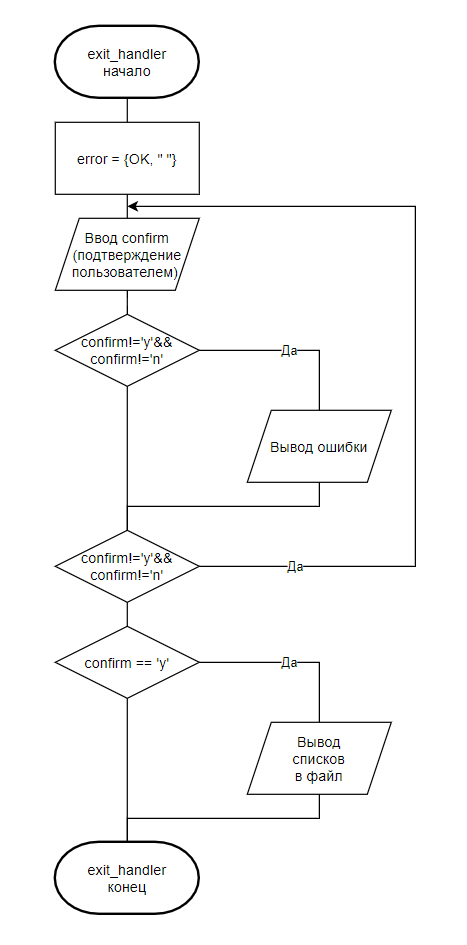
specifier\_to\_enum

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | specifier\_enum | COURSE\_TASK\_ SPECIFIER\_TYPE | Спецификатор перечислений |
| 2 | specifier | char[3] | Спецификатор |
| 3 | i | size\_t | Переменная для цикла for |

**Схема алгоритма.**

­­ 

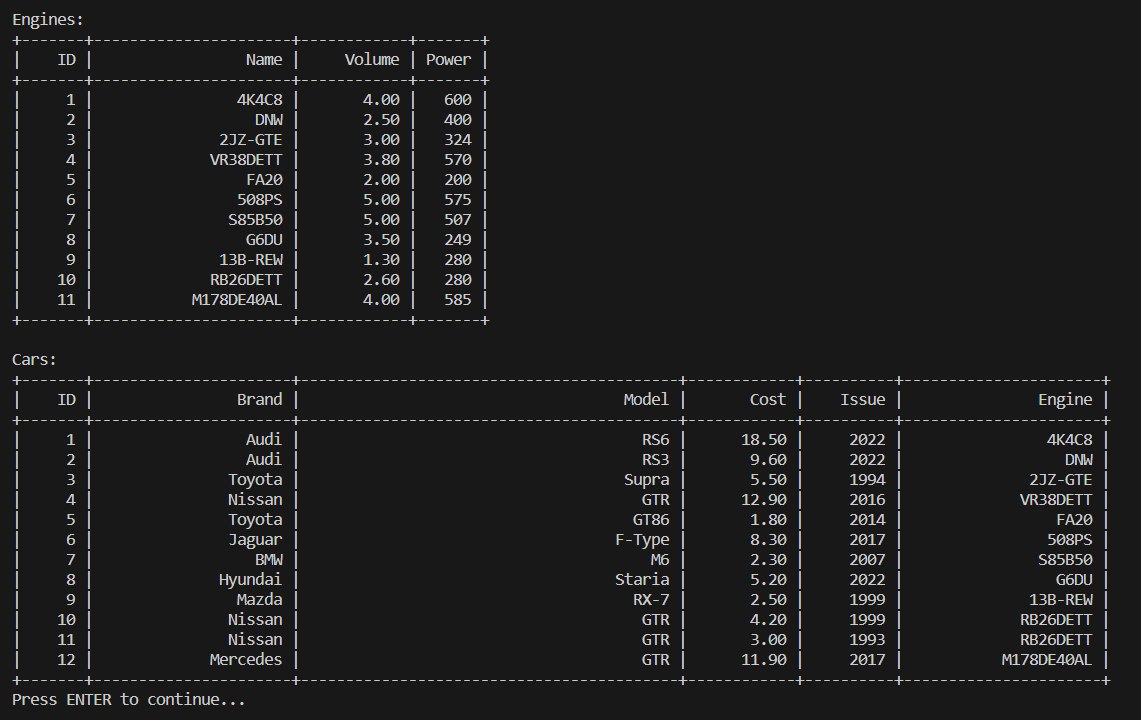
             

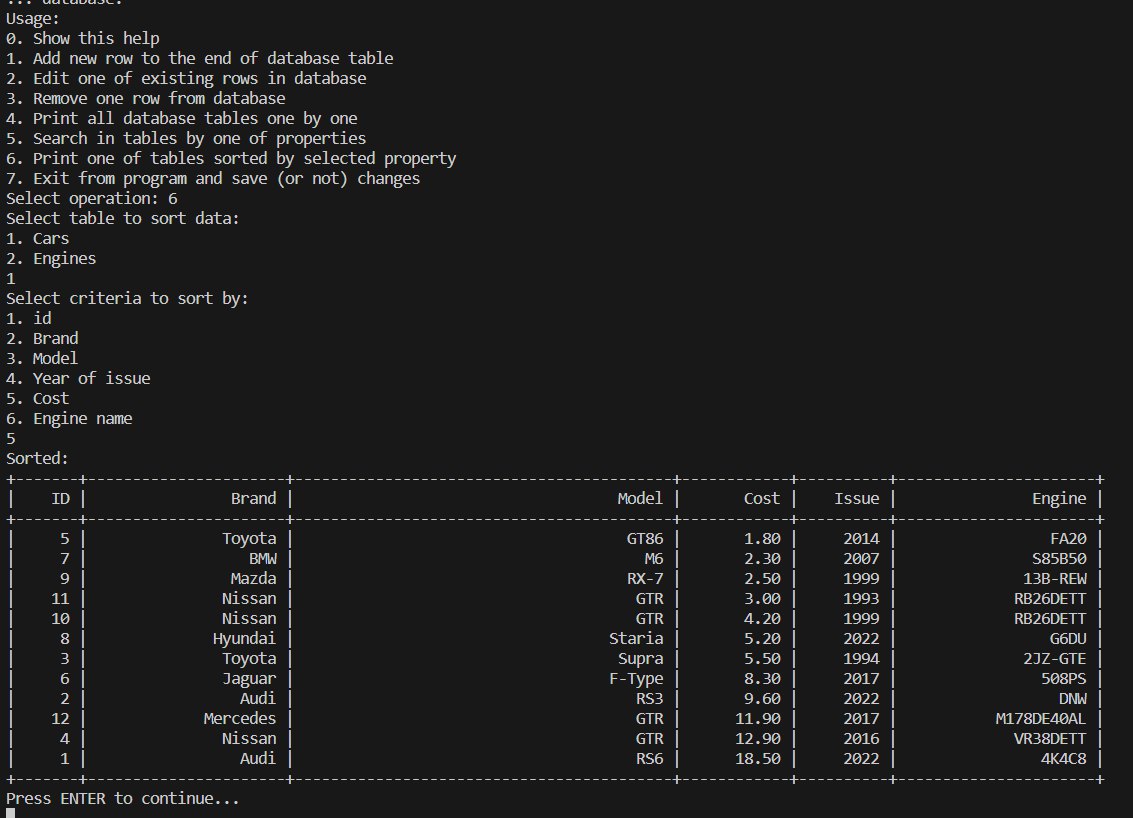
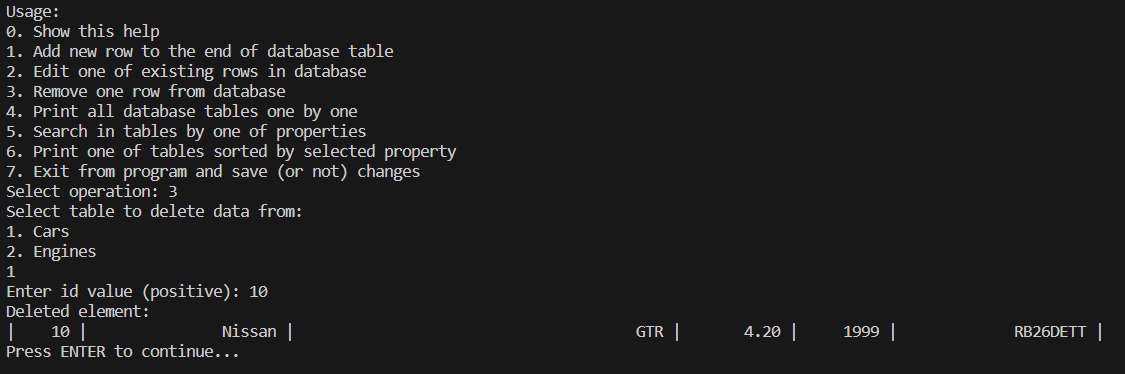
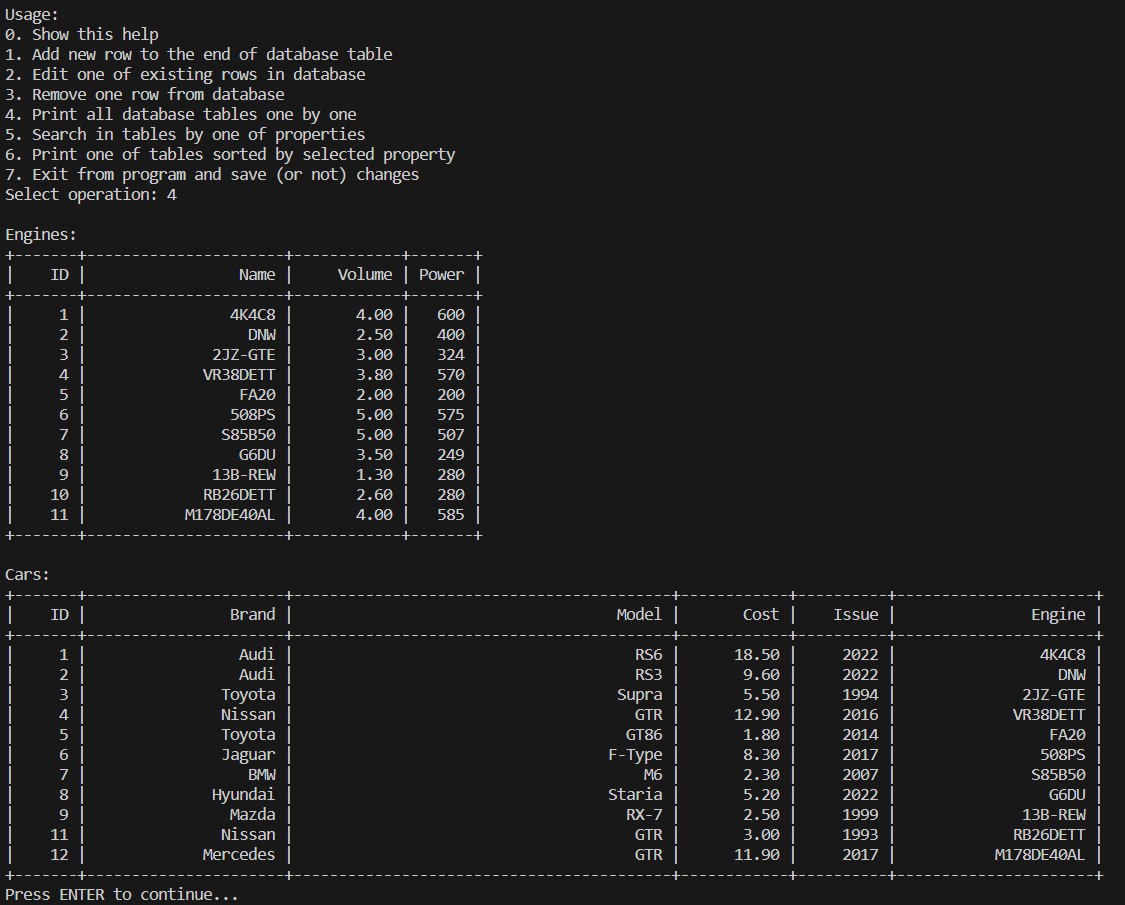
     

**Текст программы.**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include "include/csv\_file.h"  #include "include/csv\_file\_to\_list.h"  #include "include/csv\_row\_converters/csv\_row\_converters.h"  #include "include/enums.h"  #include "include/menu\_functions.h"  #include "include/row\_to\_struct.h"  #include "include/structs.h"  #include "include/utils.h"  Error main\_menu(ListHead \*cars, ListHead \*engines);  int main() {  Error error = {OK, ""};  ListHead engines = {0, NULL, NULL}, cars = {0, NULL, NULL};  /\* Database reading \*/  if (error.status\_code == OK) {  error = csv\_file\_read(  ENGINE\_FILENAME, &engines, sizeof(Engine), next\_engine, free\_engine,  (Error(\*)(CSVFileLine, void \*output, void \*))row\_to\_engine, NULL);  }  if (error.status\_code == OK) {  error = csv\_file\_read(CAR\_FILENAME, &cars, sizeof(Car), next\_car,  free\_car, row\_to\_car, &engines);  }  if (error.status\_code == OK) {  print\_data(cars, engines);  error = main\_menu(&cars, &engines);  }  switch (error.status\_code) {  case OK:  break;  case MEMORY\_ALLOCATION\_ERROR:  fputs("Error in memory alocation: ", stderr);  fputs(error.info, stderr);  fputc('\n', stderr);  break;  case FILE\_OPEN\_ERROR:  case FILE\_READ\_ERROR:  case UNKNOWN:  case PARSE\_ERROR:  fputs(error.info, stderr);  fputc('\n', stderr);  break;  }  free\_list(&cars, free\_car, next\_car);  free\_list(&engines, free\_engine, next\_engine);  return 0;  }  Error main\_menu(ListHead \*cars, ListHead \*engines) {  MenuSelection menu\_selection;  Error error = {OK, ""};  do {  about();  fputs("Select operation: ", stdout);  scanf("%d", (int \*)&menu\_selection);  fgetc(stdin);  switch (menu\_selection) {  case ABOUT:  break;  case ADD:  error = add\_new\_data(cars, engines);  break;  case EDIT:  error = edit\_data(\*cars, \*engines);  break;  case DELETE:  error = delete\_data(cars, engines);  break;  case PRINT:  print\_data(\*cars, \*engines);  break;  case SEARCH:  error = search\_data(\*cars, \*engines);  break;  case SORT:  error = sort\_data(\*cars, \*engines);  break;  case EXIT:  error = exit\_handler(\*cars, \*engines);  break;  default:  printf("Unknown selection (%d), please retry\n",  menu\_selection);  break;  }  if (error.status\_code != OK) {  fgetc(stdin);  }  clear\_screen();  } while (menu\_selection != EXIT && error.status\_code == OK);  return error;  }  csv\_file.h  /\* Parses subset of RFC4180 csv files \*/  /\*  What is supported:  RFC4180 csv scheme with some changes:  1. Only rectangular csvs supported. All lines must have the same fields count.  2. CRLF at the end of the file is not allowed. It creates ambiguous situations  when csv contains one column that could be empty. Example:  Some text  It can be parsed as two lines with "Some text" and "" or as one line with  "Some text". Ambiguous is not a good thing, so this restriction appeared.  3. File must be opened in binary mode, so no change to line endings. The  positive side of this change is whole utf-8 table supported.  \*/  #ifndef CSV\_FILE\_H  #define CSV\_FILE\_H  #include <stdio.h>  #include <stdlib.h>  /\* Error codes \*/  #define CSV\_SUCCESS 0  #define CSV\_EPARSE 1  #define CSV\_ENOMEM 2  /\* Main symbols \*/  #define CSV\_COMMA 0x2C  #define CSV\_CR 0x0d  #define CSV\_DQUOTE 0x22  #define CSV\_LF 0x0A  typedef struct CSVFileLine {  char \*\*fields;  } CSVFileLine;  typedef struct CSVFile {  size\_t fields\_count;  size\_t lines\_count;  CSVFileLine \*lines;  } CSVFile;  /\* Frees file lines based on lines\_count value \*/  void CSVFileFree(CSVFile \*file);  /\* Reads csv data from file fp to output csv struct \*/  int CSVFileRead(FILE \*fp, CSVFile \*output);  /\* Output file must be opened in binary mode \*/  void CSVFileWrite(FILE \*fp, CSVFile data);  /\* Free individual file line \*/  void \_CSVFileFreeLine(CSVFileLine \*line, size\_t fields\_count);  /\* Read one field from file \*/  int \_CSVFileReadField(FILE \*csv\_file, char \*\*output);  /\* Skip line separator in file \*/  int \_CSVFileSkipCRLF(FILE \*csv\_file);  /\* Read one line from file \*/  int \_CSVReadLine(FILE \*csv\_fiel, CSVFileLine \*output, size\_t fields\_count);  /\* Count fields in current line of file \*/  int \_CSVFileCountFields(FILE \*csv\_file, size\_t \*fields\_count);  /\* Free memeory allocated for CSVFile file and it's rows\*/  void CSVFileFree(CSVFile \*file) {  size\_t i;  if (file->lines\_count) {  for (i = 0; i < file->lines\_count; i++) {  \_CSVFileFreeLine(&file->lines[i], file->fields\_count);  }  free(file->lines);  }  file->lines = NULL;  file->lines\_count = 0;  }  /\* Read file fp contents to CSVFile output. fp MUST be opened in binary mode and  has CRLF line endings \*/  int CSVFileRead(FILE \*fp, CSVFile \*output) {  int error = CSV\_SUCCESS, continue\_loop = 1;  size\_t allocated = 0;  CSVFileLine \*realloc\_result;  char next\_char;  output->lines\_count = 0;  output->fields\_count = 0;  output->lines = NULL;  error = \_CSVFileCountFields(fp, &output->fields\_count);  continue\_loop = error == CSV\_SUCCESS;  if (error != CSV\_SUCCESS) {  puts("Error on counting fields");  }  while (continue\_loop) {  if (output->lines\_count + 1 > allocated) {  allocated = allocated ? allocated \* 2 : 1;  realloc\_result = (CSVFileLine \*)realloc(  output->lines, allocated \* sizeof(CSVFileLine));  if (realloc\_result) {  output->lines = realloc\_result;  } else {  error = CSV\_ENOMEM;  }  }  if (error == CSV\_SUCCESS) {  error = \_CSVReadLine(fp, &output->lines[output->lines\_count],  output->fields\_count);  }  if (error == CSV\_SUCCESS) {  output->lines\_count++;  /\* Skip line separator \*/  if (\_CSVFileSkipCRLF(fp) == CSV\_EPARSE) {  next\_char = fgetc(fp);  if (next\_char == EOF) {  error = CSV\_SUCCESS;  continue\_loop = 0;  } else {  error = CSV\_EPARSE;  }  }  }  continue\_loop = continue\_loop && error == CSV\_SUCCESS;  }  if (error != CSV\_SUCCESS) {  for (; output->lines\_count > 0; output->lines\_count--) {  \_CSVFileFreeLine(&output->lines[output->lines\_count - 1],  output->fields\_count);  }  free(output->lines);  output->lines = NULL;  }  return error;  }  /\* Saves CSVFile data to fp. fp MUST be opened in binary mode \*/  void CSVFileWrite(FILE \*fp, CSVFile data) {  size\_t i, j;  char CRLF[] = {CSV\_CR, CSV\_LF}, \*ch;  for (i = 0; i < data.lines\_count; i++) {  for (j = 0; j < data.fields\_count; j++) {  fputc(CSV\_DQUOTE, fp);  ch = data.lines[i].fields[j];  while (\*ch) {  fputc(\*ch, fp);  if (\*ch == CSV\_DQUOTE) {  fputc(\*ch, fp);  }  ch++;  }  fputc(CSV\_DQUOTE, fp);  if (j != data.fields\_count - 1) {  fputc(CSV\_COMMA, fp);  }  }  if (i != data.lines\_count - 1) {  fwrite(CRLF, sizeof(\*CRLF), sizeof(CRLF) / sizeof(\*CRLF), fp);  }  }  }  /\* Free single line of CSVFile \*/  void \_CSVFileFreeLine(CSVFileLine \*line, size\_t fields\_count) {  size\_t i;  for (i = 0; i < fields\_count; i++) {  free(line->fields[i]);  }  free(line->fields);  line->fields = NULL;  }  /\* Adds character to the end of string (overwrites it's 0 terminator). Used to  control allocated size of string and realloc if needed. \*/  int \_CSVFileAddCharToOutput(char character, char \*\*output, size\_t \*string\_len,  size\_t \*allocated) {  int error = CSV\_SUCCESS;  char \*realloc\_result;  (\*string\_len)++;  if (\*string\_len + 1 > \*allocated) {  \*allocated \*= 2;  realloc\_result = (char \*)realloc(\*output, sizeof(char) \* \*allocated);  if (realloc\_result) {  \*output = realloc\_result;  (\*output)[\*string\_len - 1] = character;  } else {  free(\*output);  \*output = NULL;  error = CSV\_ENOMEM;  }  } else {  (\*output)[\*string\_len - 1] = character;  }  return error;  }  /\* Read one csv field \*/  int \_CSVFileReadField(FILE \*csv\_file, char \*\*output) {  size\_t string\_len = 0, allocated = 1, i;  char quoted = 0, continue\_loop = 1;  char \*realloc\_result = NULL;  char current\_char;  int error = CSV\_SUCCESS;  \*output = (char \*)malloc(sizeof(char) \* allocated);  /\* Test if it's quoted field and read first\*/  if ((current\_char = fgetc(csv\_file)) != EOF) {  if (current\_char == CSV\_DQUOTE) {  quoted = 1;  } else {  fseek(csv\_file, -1, SEEK\_CUR);  }  }  continue\_loop = current\_char != EOF;  while (continue\_loop) {  current\_char = fgetc(csv\_file);  if (quoted) {  switch (current\_char) {  case CSV\_DQUOTE:  i = 1;  while ((current\_char = fgetc(csv\_file)) == CSV\_DQUOTE &&  error == CSV\_SUCCESS) {  if (++i % 2 == 0) {  error = \_CSVFileAddCharToOutput(  current\_char, output, &string\_len, &allocated);  }  }  if (error == CSV\_SUCCESS) {  if (current\_char == EOF) {  continue\_loop = 0;  } else {  if (i % 2) {  if (current\_char == CSV\_COMMA ||  current\_char == CSV\_CR) {  continue\_loop = 0;  } else {  error = CSV\_EPARSE;  }  } else {  fseek(csv\_file, -1, SEEK\_CUR);  }  }  }  break;  case EOF:  error = CSV\_EPARSE;  break;  default:  error = \_CSVFileAddCharToOutput(current\_char, output,  &string\_len, &allocated);  }  } else {  switch (current\_char) {  case CSV\_COMMA:  case CSV\_CR:  case EOF:  continue\_loop = 0;  break;  case CSV\_DQUOTE:  error = CSV\_EPARSE;  break;  default:  error = \_CSVFileAddCharToOutput(current\_char, output,  &string\_len, &allocated);  break;  }  }  continue\_loop = continue\_loop && error == CSV\_SUCCESS;  }  switch (error) {  case CSV\_EPARSE:  free(\*output);  \*output = NULL;  break;  case CSV\_SUCCESS:  (\*output)[string\_len] = 0;  realloc\_result =  (char \*)realloc(\*output, (string\_len + 1) \* sizeof(char));  if (realloc\_result) {  \*output = realloc\_result;  }  if (current\_char != EOF) {  /\* Go to separator symbol \*/  fseek(csv\_file, -1, SEEK\_CUR);  }  break;  }  return error;  }  /\* Count fields in file by first line info \*/  int \_CSVFileCountFields(FILE \*csv\_file, size\_t \*fields\_count) {  long start = ftell(csv\_file);  char quoted = 0, current\_char, previous\_char = 0, continue\_loop = 1;  int error = CSV\_SUCCESS;  size\_t quotes\_count;  \*fields\_count = 0;  while (continue\_loop) {  current\_char = fgetc(csv\_file);  switch (current\_char) {  case EOF:  if (quoted) {  error = CSV\_EPARSE;  } else {  (\*fields\_count)++;  continue\_loop = 0;  }  break;  case CSV\_COMMA:  if (!quoted) {  (\*fields\_count)++;  }  break;  case CSV\_CR:  if (!quoted) {  (\*fields\_count)++;  continue\_loop = 0;  }  break;  case CSV\_DQUOTE:  quotes\_count = 1;  while ((current\_char = fgetc(csv\_file)) == CSV\_DQUOTE) {  quotes\_count++;  }  if (current\_char != EOF) {  fseek(csv\_file, -1, SEEK\_CUR);  }  if (quotes\_count % 2) {  if (quoted) {  quoted = 0;  } else {  if (previous\_char == CSV\_COMMA || previous\_char == 0) {  quoted = 1;  } else {  error = CSV\_EPARSE;  }  }  }  break;  }  previous\_char = current\_char;  continue\_loop = continue\_loop && error == CSV\_SUCCESS;  }  if (error != CSV\_SUCCESS) {  \*fields\_count = 0;  }  fseek(csv\_file, start, SEEK\_SET);  return error;  }  /\* Skip line ending with strict characters check \*/  int \_CSVFileSkipCRLF(FILE \*csv\_file) {  int error = CSV\_SUCCESS;  char next;  if ((next = fgetc(csv\_file)) == CSV\_CR) {  if ((next = fgetc(csv\_file)) == CSV\_LF) {  } else {  error = CSV\_EPARSE;  if (next == EOF) {  fseek(csv\_file, -1, SEEK\_CUR);  } else {  fseek(csv\_file, -2, SEEK\_CUR);  }  }  } else {  error = CSV\_EPARSE;  if (next != EOF) {  fseek(csv\_file, -1, SEEK\_CUR);  }  }  return error;  }  /\* Read one csv line \*/  int \_CSVReadLine(FILE \*csv\_file, CSVFileLine \*output, size\_t fields\_count) {  int error = CSV\_SUCCESS;  size\_t i;  char continue\_loop = 1, next\_char;  output->fields = (char \*\*)malloc(sizeof(char \*) \* fields\_count);  if (!output->fields) {  error = CSV\_ENOMEM;  continue\_loop = 0;  }  i = 0;  while (continue\_loop) {  if (i < fields\_count) {  error = \_CSVFileReadField(csv\_file, &output->fields[i]);  if (error == CSV\_SUCCESS) {  next\_char = fgetc(csv\_file); /\* skip separator \*/  if (next\_char == EOF) {  continue\_loop = 0;  } else {  if (next\_char == CSV\_CR) {  continue\_loop = 0;  fseek(csv\_file, -1, SEEK\_CUR);  }  }  i += 1;  }  } else {  error = CSV\_EPARSE;  puts("Out of fields bounds!");  }  continue\_loop = continue\_loop && error == CSV\_SUCCESS;  }  /\* If read less than needed, than it's wrong \*/  if (error == CSV\_SUCCESS) {  if (i != fields\_count) {  error = CSV\_EPARSE;  }  }  if (error != CSV\_SUCCESS) {  for (; i > 0; i--) {  free(output->fields[i - 1]);  }  free(output->fields);  output->fields = NULL;  }  return error;  }  #endif  csv\_file\_to\_list.h  #ifndef COURSE\_TASK\_CSV\_FILE\_TO\_LIST  #define COURSE\_TASK\_CSV\_FILE\_TO\_LIST  #include <stdlib.h>  #include "csv\_file.h"  #include "enums.h"  #include "row\_to\_struct.h"  #include "structs.h"  /\* Converts CSVFile csv to list by converting each row of csv via row\_converter  function. free\_func is used if error occured.  Element size passed to malloc, so it must be sizeof(ElementType) \*/  Error csv\_file\_to\_list(CSVFile csv, ListHead \*output, size\_t element\_size,  void \*\*(\*next\_getter)(void \*),  void (\*free\_func)(void \*\*),  Error (\*row\_converter)(CSVFileLine, void \*, void \*),  void \*additional\_args) {  Error error = {OK, ""};  size\_t i;  output->first = NULL;  output->last = NULL;  output->count = 0;  for (i = 0; i < csv.lines\_count && error.status\_code == OK; i++) {  if (output->first) {  \*next\_getter(output->last) = malloc(element\_size);  output->last = \*next\_getter(output->last);  } else {  output->last = malloc(element\_size);  output->first = output->last;  }  if (output->last) {  \*next\_getter(output->last) = NULL;  error = row\_converter(csv.lines[i], output->last, additional\_args);  } else {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info = "Can't allocate memory for csv file's lines";  }  }  if (error.status\_code == MEMORY\_ALLOCATION\_ERROR) {  free\_list(output, free\_func, next\_getter);  } else if (error.status\_code == OK) {  output->count = csv.lines\_count;  }  return error;  }  #endif  ehums.h  #ifndef COURSE\_TASK\_ENUMS  #define COURSE\_TASK\_ENUMS  #define ENGINE\_FILENAME "db/engines.csv"  #define CAR\_FILENAME "db/cars.csv"  /\* General enums \*/  typedef enum StatusCode {  OK,  MEMORY\_ALLOCATION\_ERROR,  FILE\_OPEN\_ERROR,  FILE\_READ\_ERROR,  PARSE\_ERROR,  UNKNOWN  } StatusCode;  typedef enum MenuSelection {  ABOUT,  ADD,  EDIT,  DELETE,  PRINT,  SEARCH,  SORT,  EXIT  } MenuSelection;  typedef enum VariableType {  CHAR,  DOUBLE,  FLOAT,  INT,  STRING,  UNSIGNED  } VariableType;  #endif  getters.h  #ifndef COURSE\_TASK\_GETTERS  #define COURSE\_TASK\_GETTERS  #include "structs.h"  /\* Getters for created structures \*/  /\* Car field getters\*/  unsigned \*car\_get\_engine\_id(void \*car) { return &((Car \*)car)->engine->id; }  char \*\*car\_get\_engine\_name(void \*car) { return &((Car \*)car)->engine->name; }  float \*car\_get\_cost(void \*car) { return &((Car \*)car)->cost; }  char \*\*car\_get\_model(void \*car) { return &((Car \*)car)->model; }  unsigned \*car\_get\_id(void \*car) { return &((Car \*)car)->id; }  char \*\*car\_get\_brand(void \*car) { return &((Car \*)car)->brand; }  int \*car\_get\_year\_of\_issue(void \*car) { return &((Car \*)car)->year\_of\_issue; }  void \*\*next\_car(void \*car) { return (void \*\*)&((Car \*)car)->next; }  /\* Engine field getters\*/  unsigned \*engine\_get\_id(void \*engine) { return &((Engine \*)engine)->id; }  char \*\*engine\_get\_name(void \*engine) { return &((Engine \*)engine)->name; }  float \*engine\_get\_volume(void \*engine) { return &((Engine \*)engine)->volume; }  int \*engine\_get\_power(void \*engine) { return &((Engine \*)engine)->power; }  void \*\*next\_engine(void \*engine) { return (void \*\*)&((Engine \*)engine)->next; }  /\* ListElement field getters \*/  void \*\*list\_element\_get\_data(void \*list\_element) {  return (void \*\*)&((ListElement \*)list\_element)->data;  }  void \*\*next\_list\_element(void \*list\_element) {  return (void \*\*)&((ListElement \*)list\_element)->next;  }  #endif  menu\_functions.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS  #define COURSE\_TASK\_MENU\_FUNCTIONS  #include <stdio.h>  #include <stdlib.h>  #include "csv\_file.h"  #include "enums.h"  #include "getters.h"  #include "menu\_functions/add\_funcs.h"  #include "menu\_functions/edit\_funcs.h"  #include "menu\_functions/exit\_funcs.h"  #include "menu\_functions/input\_funcs.h"  #include "menu\_functions/search\_funcs.h"  #include "menu\_functions/sort\_funcs.h"  #include "menu\_functions/utils.h"  #include "printers.h"  #include "row\_to\_struct.h"  #include "struct\_to\_row.h"  #include "structs.h"  #include "utils.h"  /\* Force user to select one of variants from min\_value up to max\_value inclusive  \*/  int select\_prompt(char \*prompt, int min\_value, int max\_value);  /\* Shows short help with program usages \*/  void about() {  printf(  "... database.\n"  "Usage:\n"  "%d. Show this help\n"  "%d. Add new row to the end of database table\n"  "%d. Edit one of existing rows in database\n"  "%d. Remove one row from database\n"  "%d. Print all database tables one by one\n"  "%d. Search in tables by one of properties\n"  "%d. Print one of tables sorted by selected property\n"  "%d. Exit from program and save (or not) changes\n",  ABOUT, ADD, EDIT, DELETE, PRINT, SEARCH, SORT, EXIT);  }  /\* Add new data to database \*/  Error add\_new\_data(ListHead \*cars, ListHead \*engines) {  Error error = {OK, ""};  int table\_selection = -1;  table\_selection = select\_prompt(  "Select table to add element to:\n1. Cars\n2. Engines", 1, 2);  switch (table\_selection) {  case 1:  error = add\_car(cars, \*engines);  break;  case 2:  error = add\_engine(engines);  break;  }  return error;  }  /\* Edit data in database \*/  Error edit\_data(ListHead cars, ListHead engines) {  Error error = {OK, ""};  int table\_selection;  void \*element, \*replacement;  table\_selection =  select\_prompt("Select table to edit data:\n1. Cars\n2. Engines", 1, 2);  switch (table\_selection) {  case 1:  print\_cars\_list(cars);  element = select\_element(cars, car\_get\_id, next\_car, print\_car);  if (element) {  replacement = input\_car(engines);  \*car\_get\_id(replacement) = \*car\_get\_id(element);  ((Car \*)replacement)->next = ((Car \*)element)->next;  free(((Car \*)element)->brand);  free(((Car \*)element)->model);  \*((Car \*)element) = \*((Car \*)replacement);  }  break;  case 2:  print\_engines\_list(engines);  element = select\_element(engines, engine\_get\_id, next\_engine,  print\_engine);  if (element) {  replacement = input\_engine();  \*engine\_get\_id(replacement) = \*engine\_get\_id(element);  ((Engine \*)replacement)->next = ((Engine \*)element)->next;  free(((Engine \*)element)->name);  \*((Engine \*)element) = \*((Engine \*)replacement);  }  break;  }  if (replacement) {  free(replacement);  }  return error;  }  /\* Delete data from database \*/  Error delete\_data(ListHead \*cars, ListHead \*engines) {  Error error = {OK, ""};  int table\_selection = -1;  unsigned id;  void \*iter;  char found = 1;  table\_selection = select\_prompt(  "Select table to delete data from:\n1. Cars\n2. Engines", 1, 2);  fputs("Enter id value (positive): ", stdout);  scanf("%u", &id);  fgetc(stdin);  switch (table\_selection) {  case 1:  iter = find\_element\_with\_id(\*cars, id, car\_get\_id, next\_car);  if (iter) {  puts("Deleted element:");  print\_car((Car \*)iter);  delete\_list\_element(cars, iter, next\_car, free\_car);  } else {  iter = 0;  }  break;  case 2:  if ((iter = find\_element\_with\_id(\*cars, id, car\_get\_engine\_id,  next\_car))) {  printf(  "Can't delete engine with id=%u because "  "car with id=%u depends on it\n",  id, ((Car \*)iter)->id);  } else {  iter = find\_element\_with\_id(\*engines, id, engine\_get\_id,  next\_engine);  if (iter) {  puts("Deleted element:");  print\_engine((Engine \*)iter);  delete\_list\_element(engines, iter, next\_engine,  free\_engine);  } else {  found = 0;  }  }  break;  }  if (!found) {  printf("Element with id %u not found!\n", id);  }  puts("Press ENTER to continue...");  fgetc(stdin);  return error;  }  /\* Print whole database. One table after another \*/  void print\_data(const ListHead cars, const ListHead engines) {  puts("\nEngines:");  print\_engines\_list(engines);  puts("\nCars:");  print\_cars\_list(cars);  puts("Press ENTER to continue...");  fgetc(stdin);  }  /\* Search in database by user selected field \*/  Error search\_data(ListHead cars, ListHead engines) {  Error error = {OK, ""};  int table\_selection;  int search\_criteria;  static void \*(\*car\_field\_getters[])(void \*) = {  (void \*(\*)(void \*))car\_get\_id,  (void \*(\*)(void \*))car\_get\_brand,  (void \*(\*)(void \*))car\_get\_model,  (void \*(\*)(void \*))car\_get\_year\_of\_issue,  (void \*(\*)(void \*))car\_get\_cost,  (void \*(\*)(void \*))car\_get\_engine\_name,  };  static void \*(\*engine\_field\_getters[])(void \*) = {  (void \*(\*)(void \*))engine\_get\_id,  (void \*(\*)(void \*))engine\_get\_name,  (void \*(\*)(void \*))engine\_get\_volume,  (void \*(\*)(void \*))engine\_get\_power,  };  char buf[1024];  size\_t value\_size;  char is\_str, found;  table\_selection = select\_prompt(  "Select table to search data in:\n1. Cars\n2. Engines", 1, 2);  switch (table\_selection) {  case 1:  search\_criteria = select\_prompt(  "Select criteria to search by:"  "\n1. id"  "\n2. Brand"  "\n3. Model"  "\n4. Year of issue"  "\n5. Cost"  "\n6. Engine name",  1, 6);  fputs("Enter value:", stdout);  switch (search\_criteria) {  case 1:  scanf("%u", (unsigned \*)buf);  fgetc(stdin);  value\_size = sizeof(unsigned);  is\_str = 0;  break;  case 2:  case 3:  case 6:  fgets(buf, sizeof(buf), stdin);  remove\_last\_symbol(buf);  is\_str = 1;  break;  case 4:  scanf("%d", (int \*)buf);  fgetc(stdin);  value\_size = sizeof(int);  is\_str = 0;  break;  case 5:  scanf("%f", (float \*)buf);  fgetc(stdin);  value\_size = sizeof(float);  is\_str = 0;  break;  }  found = search(cars, buf, value\_size,  car\_field\_getters[search\_criteria - 1], next\_car,  print\_car, is\_str);  break;  case 2:  search\_criteria = select\_prompt(  "Select criteria to search by:"  "\n1. id"  "\n2. Name"  "\n3. Volume"  "\n4. Power",  1, 4);  fputs("Enter value:", stdout);  switch (search\_criteria) {  case 1:  scanf("%u", (unsigned \*)buf);  fgetc(stdin);  is\_str = 0;  value\_size = sizeof(unsigned);  break;  case 2:  fgets(buf, sizeof(buf), stdin);  remove\_last\_symbol(buf);  is\_str = 1;  break;  case 3:  scanf("%f", (float \*)buf);  fgetc(stdin);  is\_str = 0;  value\_size = sizeof(float);  break;  case 4:  scanf("%d", (int \*)buf);  fgetc(stdin);  value\_size = sizeof(int);  is\_str = 0;  }  found = search(engines, buf, sizeof(unsigned),  engine\_field\_getters[search\_criteria - 1],  next\_engine, print\_engine, is\_str);  break;  }  if (!found) {  puts("Elements that meet such criteria not found");  }  puts("Press ENTER to continue...");  fgetc(stdin);  return error;  }  /\* Sort database data and print sorted. Than return data to previous state \*/  Error sort\_data(ListHead cars, ListHead engines) {  Error error = {OK, ""};  int table\_selection, sort\_criteria;  VariableType sort\_type;  ListHead sorted = {0, NULL, NULL};  ListElement \*element;  static void \*(\*car\_field\_getters[])(void \*) = {  (void \*(\*)(void \*))car\_get\_id,  (void \*(\*)(void \*))car\_get\_brand,  (void \*(\*)(void \*))car\_get\_model,  (void \*(\*)(void \*))car\_get\_year\_of\_issue,  (void \*(\*)(void \*))car\_get\_cost,  (void \*(\*)(void \*))car\_get\_engine\_name,  };  static void \*(\*engine\_field\_getters[])(void \*) = {  (void \*(\*)(void \*))engine\_get\_id,  (void \*(\*)(void \*))engine\_get\_name,  (void \*(\*)(void \*))engine\_get\_volume,  (void \*(\*)(void \*))engine\_get\_power,  };  void (\*printer)(const void \*);  void (\*separator\_printer)();  void (\*header\_printer)();  table\_selection = select\_prompt(  "Select table to sort data:\n1. Cars\n2. "  "Engines",  1, 2);  switch (table\_selection) {  case 1:  printer = print\_car;  separator\_printer = print\_cars\_list\_separator;  header\_printer = print\_cars\_list\_header;  sort\_criteria = select\_prompt(  "Select criteria to sort by:"  "\n1. id"  "\n2. Brand"  "\n3. Model"  "\n4. Year of issue"  "\n5. Cost"  "\n6. Engine name",  1, 6);  switch (sort\_criteria) {  case 1:  sort\_type = UNSIGNED;  break;  case 5:  sort\_type = INT;  break;  case 2:  case 3:  case 6:  sort\_type = STRING;  break;  case 4:  sort\_type = FLOAT;  break;  }  error =  sort\_list(cars, &sorted, car\_field\_getters[sort\_criteria - 1],  next\_car, sort\_type);  break;  case 2:  printer = print\_engine;  separator\_printer = print\_engines\_list\_separator;  header\_printer = print\_engines\_list\_header;  sort\_criteria = select\_prompt(  "Select criteria to sort by:"  "\n1. id"  "\n2. Name"  "\n3. Volume"  "\n4. Power",  1, 4);  switch (sort\_criteria) {  case 1:  sort\_type = UNSIGNED;  break;  case 2:  sort\_type = STRING;  break;  case 3:  sort\_type = FLOAT;  break;  case 4:  sort\_type = INT;  break;  }  error = sort\_list(engines, &sorted,  engine\_field\_getters[sort\_criteria - 1],  next\_engine, sort\_type);  break;  }  if (error.status\_code == OK) {  puts("Sorted:");  element = (ListElement \*)sorted.first;  separator\_printer();  header\_printer();  separator\_printer();  while (element) {  printer(\*list\_element\_get\_data(element));  element = (ListElement \*)\*next\_list\_element(element);  }  separator\_printer();  }  free\_list(&sorted, free\_list\_element, next\_list\_element);  puts("Press ENTER to continue...");  fgetc(stdin);  return error;  }  /\* Exit from program and save data to database if user wants\*/  Error exit\_handler(ListHead cars, ListHead engines) {  Error error = {OK, ""};  char confirm;  do {  fputs("Save changes (y/n): ", stdout);  confirm = fgetc(stdin);  fgetc(stdin);  if (confirm != 'y' && confirm != 'n') {  printf("Unknown selection: %c. Please, retry.\n", confirm);  }  } while (confirm != 'y' && confirm != 'n');  if (confirm == 'y') {  error = save\_list\_to\_file(engines, ENGINE\_FILENAME, 4, engine\_to\_row,  next\_engine);  if (error.status\_code == OK) {  error =  save\_list\_to\_file(cars, CAR\_FILENAME, 6, car\_to\_row, next\_car);  }  }  return error;  }  int select\_prompt(char \*prompt, int min\_value, int max\_value) {  int table\_selection;  do {  puts(prompt);  scanf("%d", &table\_selection);  fgetc(stdin);  if (table\_selection < min\_value || table\_selection > max\_value) {  printf("Wrong selection: %d. Please, retry\n", table\_selection);  }  } while (table\_selection < min\_value || table\_selection > max\_value);  return table\_selection;  }  #endif  printers.h  #ifndef COURSE\_TASK\_PRINTERS  #define COURSE\_TASK\_PRINTERS  #include <stdio.h>  #include "structs.h"  /\* Print single car \*/  void print\_car(const void \*car) {  const Car \*car\_struct = car;  printf("| %5u | %20s | %40s | %10.2f | %8d | %20s |\n", car\_struct->id,  car\_struct->brand, car\_struct->model, car\_struct->cost,  car\_struct->year\_of\_issue, car\_struct->engine->name);  }  /\* Print cars list separator. Also used as table top and bottom borders \*/  void print\_cars\_list\_separator() {  puts(  "+-------+----------------------+--------------------------------------"  "----+------------+----------+----------------------+");  }  /\* Print cars table header \*/  void print\_cars\_list\_header() {  printf("| %5s | %20s | %40s | %10s | %8s | %20s |\n", "ID", "Brand",  "Model", "Cost", "Issue", "Engine");  }  /\* Print list of cars \*/  void print\_cars\_list(const ListHead cars) {  Car \*car = cars.first;  print\_cars\_list\_separator();  print\_cars\_list\_header();  print\_cars\_list\_separator();  while (car) {  print\_car(car);  car = car->next;  }  print\_cars\_list\_separator();  }  /\* Print single engine \*/  void print\_engine(const void \*engine) {  const Engine \*engine\_struct = engine;  printf("| %5u | %20s | %10.2f | %5d |\n", engine\_struct->id,  engine\_struct->name, engine\_struct->volume, engine\_struct->power);  }  /\* Print engines list separator. Also used as table top and bottom borders \*/  void print\_engines\_list\_separator() {  puts("+-------+----------------------+------------+-------+");  }  /\* Print engines table header \*/  void print\_engines\_list\_header() {  printf("| %5s | %20s | %10s | %5s |\n", "ID", "Name", "Volume", "Power");  }  /\* Print engines table \*/  void print\_engines\_list(const ListHead engines) {  Engine \*engine = engines.first;  print\_engines\_list\_separator();  print\_engines\_list\_header();  print\_engines\_list\_separator();  while (engine) {  print\_engine(engine);  engine = engine->next;  }  print\_engines\_list\_separator();  }  #endif  row\_to\_struct.h  #ifndef COURSE\_TASK\_ROW\_TO\_STRUCT  #define COURSE\_TASK\_ROW\_TO\_STRUCT  #include <stdarg.h>  #include "csv\_file.h"  #include "csv\_row\_converters/csv\_row\_converters.h"  #include "enums.h"  #include "structs.h"  /\* Converts CSVFileLine to Engine structure \*/  Error row\_to\_engine(CSVFileLine row, void \*engine, ...) {  Engine \*pseudo = engine;  return scan\_row(row, "%u%s%f%d", &pseudo->id, &pseudo->name,  &pseudo->volume, &pseudo->power);  }  /\* Converts CSVFileLine to Car structure \*/  Error row\_to\_car(CSVFileLine row, void \*car, void \*additional\_args) {  ListHead engines;  unsigned engine\_id;  Engine \*engine = NULL;  Error error;  Car \*pseudo = car;  engines = \*(ListHead \*)additional\_args;  error = scan\_row(row, "%u%s%s%d%f%u", &pseudo->id, &pseudo->brand,  &pseudo->model, &pseudo->year\_of\_issue, &pseudo->cost,  &engine\_id);  if (error.status\_code == OK) {  pseudo->engine = NULL;  engine = (Engine \*)engines.first;  while (engine) {  if (engine->id == engine\_id) {  pseudo->engine = engine;  engine = NULL;  } else {  engine = engine->next;  }  }  if (pseudo->engine == NULL) {  error.status\_code = PARSE\_ERROR;  error.info = "Engine id not found in database";  }  }  return error;  }  #endif  structs\_to\_row.h  #ifndef COURSE\_TASK\_STRUCT\_TO\_ROW  #define COURSE\_TASK\_STRUCT\_TO\_ROW  #include "csv\_row\_converters/csv\_row\_converters.h"  #include "enums.h"  #include "structs.h"  /\* Converts Car to CSVFileLine \*/  Error car\_to\_row(void \*car, CSVFileLine \*row) {  Car \*car\_struct = (Car \*)car;  return create\_row(row, "%u%s%s%d%f%u", car\_struct->id, car\_struct->brand,  car\_struct->model, car\_struct->year\_of\_issue,  car\_struct->cost, car\_struct->engine->id);  }  /\* Converts Engine to CSVFileLine \*/  Error engine\_to\_row(void \*engine, CSVFileLine \*row) {  Engine \*engine\_struct = (Engine \*)engine;  return create\_row(row, "%u%s%f%d", engine\_struct->id, engine\_struct->name,  engine\_struct->volume, engine\_struct->power);  }  #endif  structs.h  #ifndef COURSE\_TASK\_STRUCTS  #define COURSE\_TASK\_STRUCTS  #include <stddef.h>  #include "enums.h"  typedef struct Error {  StatusCode status\_code;  char \*info;  } Error;  typedef struct Engine {  unsigned id;  char \*name;  float volume;  int power;  struct Engine \*next;  } Engine;  typedef struct Car {  unsigned id;  char \*brand;  char \*model;  int year\_of\_issue; /\* In days \*/  float cost;  Engine \*engine;  struct Car \*next;  } Car;  typedef struct ListHead {  size\_t count; /\* List elements count \*/  void \*first;  void \*last;  } ListHead;  typedef struct ListElement {  void \*data;  void \*next;  } ListElement;  /\* Frees Car and sets it's pointer to NULL \*/  void free\_car(void \*\*car) {  Car \*car\_struct = \*((Car \*\*)car);  if (car\_struct) {  if (car\_struct->brand) {  free(car\_struct->brand);  }  if (car\_struct->model) {  free(car\_struct->model);  }  free(car\_struct);  \*car = NULL;  }  }  /\* Frees Engine and sets it's pointer to NULL \*/  void free\_engine(void \*\*engine) {  Engine \*engine\_struct = \*((Engine \*\*)engine);  if (engine\_struct) {  if (engine\_struct->name) {  free(engine\_struct->name);  }  free(engine\_struct);  \*engine = NULL;  }  }  /\* Frees ListElement and sets it's pointer to NULL \*/  void free\_list\_element(void \*\*list\_element) {  if (\*list\_element) {  free(\*list\_element);  \*list\_element = NULL;  }  }  /\* Universal list free function. Uses free\_func to free one element and  \* next\_func to iterate over elements\*/  void free\_list(ListHead \*list, void (\*free\_func)(void \*\*),  void \*\*(\*next\_func)(void \*)) {  while ((list->last = list->first)) {  list->first = \*next\_func(list->first);  free\_func(&list->last);  }  list->first = NULL;  list->last = NULL;  list->count = 0;  }  #endif  utils.h  #ifndef COURSE\_TASK\_UTILS  #define COURSE\_TASK\_UTILS  #include <stdio.h>  #include <stdlib.h>  #include "csv\_file.h"  #include "csv\_file\_to\_list.h"  #include "enums.h"  #include "structs.h"  /\* Force usage of Linux's and MacOS clear command\*/  /\* #define FORCE\_UNIX \*/  /\* Force usage of Windows's clear command\*/  /\* #define FORCE\_WINDOWS \*/  void clear\_screen() {  #if defined(\_\_linux\_\_) || defined(FORCE\_UNIX)  system("clear");  #endif  #if defined(\_WIN32) || defined(FORCE\_WINDOWS)  system("cls");  #endif  }  /\* If file not exists, creates it, then open in binary update mode (r+b) and  \* seek start of file  \*/  FILE \*open\_or\_create\_file(char \*filename) {  StatusCode error = OK;  FILE \*f;  f = fopen(filename, "rb");  if (f) {  fclose(f);  } else {  f = fopen(filename, "wb");  if (f) {  fclose(f);  } else {  error = FILE\_OPEN\_ERROR;  }  }  if (error == OK) {  f = fopen(filename, "r+b");  fseek(f, 0, SEEK\_SET);  } else {  f = NULL;  }  return f;  }  /\* Wrapper above CSVFileRead and row\_converter. Reads file and converts it into  \* list via row\_converter function. If file not exists it creates empty file.  \* element\_size is size of one element of output, so it must be  \* sizeof(ElementType) \*/  Error csv\_file\_read(char \*filename, ListHead \*output, size\_t element\_size,  void \*\*(\*next\_getter)(void \*), void (\*free\_func)(void \*\*),  Error (\*row\_converter)(CSVFileLine, void \*output, void \*),  void \*additional\_args) {  FILE \*f;  Error error = {OK, ""};  CSVFile csv = {0, 0, NULL};  int csv\_status;  f = open\_or\_create\_file(filename);  if (f) {  csv\_status = CSVFileRead(f, &csv);  if (csv\_status != CSV\_SUCCESS) {  error.info = "Error parsing csv file";  error.status\_code = FILE\_READ\_ERROR;  }  fclose(f);  } else {  error.info = "Can't open csv file";  error.status\_code = FILE\_OPEN\_ERROR;  }  if (error.status\_code == OK) {  error = csv\_file\_to\_list(csv, output, element\_size, next\_getter,  free\_func, row\_converter, additional\_args);  CSVFileFree(&csv);  }  return error;  }  /\* Removes last symbol from string by setting it to 0 \*/  void remove\_last\_symbol(char \*str) {  size\_t len = strlen(str);  if (len > 0) {  str[len - 1] = 0;  }  }  /\* Removes leftover '\n' symbols from stdin \*/  void flush\_stdin() {  char c;  while ((c = getchar()) != '\n' && c != EOF)  ;  }  #endif  csv\_row\_converters.h  /\* VERY strange code that uses enum inside macro to not mess with global  \* namespace\*/  #ifndef COURSE\_TASK\_CSV\_ROW\_CONVERTERS  #define COURSE\_TASK\_CSV\_ROW\_CONVERTERS  #include <float.h>  #include <limits.h>  #include <math.h>  #include <stdarg.h>  #include <string.h>  #include "../csv\_file.h"  #include "../enums.h"  #include "../structs.h"  #include "utils.h"  /\* Use standard format specifiers like %s, %c %d etc.  \* Format example: "%s%d%c%d%d".  \* This function doesn't check fields count.  \* Supported specifiers:  \* c  \* d  \* e  \* f  \* g  \* hu  \* hd  \* ld  \* le  \* lf  \* lg  \* lu  \* s  \* u  \*/  /\* Works as scanf except for char\* - it needs char\*\* to auto malloc memory.  Support only type specifiers, no separators and other format specifiers — only  such that listed in first forty lines.  \*/  Error scan\_row(CSVFileLine row, const char \*format, ...) {  COURSE\_TASK\_SPECIFIER\_TYPE \*specifiers = NULL;  size\_t count, i;  Error error;  va\_list args;  char \*\*str;  int temp;  va\_start(args, format);  error = format\_str\_to\_enum\_list(format, (int \*\*)&specifiers, &count);  for (i = 0; i < count && error.status\_code == OK; i++) {  switch (specifiers[i]) {  case c:  temp = sscanf(row.fields[i], "%c", va\_arg(args, char \*));  break;  case d:  temp = sscanf(row.fields[i], "%d", va\_arg(args, int \*));  break;  case f:  temp = sscanf(row.fields[i], "%f", va\_arg(args, float \*));  break;  case hd:  temp = sscanf(row.fields[i], "%hd", va\_arg(args, short int \*));  break;  case hu:  temp = sscanf(row.fields[i], "%hu",  va\_arg(args, short unsigned \*));  break;  case ld:  temp = sscanf(row.fields[i], "%ld", va\_arg(args, long \*));  break;  case lf:  temp = sscanf(row.fields[i], "%lf", va\_arg(args, double \*));  break;  case lu:  temp =  sscanf(row.fields[i], "%lu", va\_arg(args, long unsigned \*));  break;  case s:  temp = 1;  str = va\_arg(args, char \*\*);  \*str =  (char \*)malloc(sizeof(char) \* (strlen(row.fields[i]) + 1));  if (\*str) {  strcpy(\*str, row.fields[i]);  } else {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info =  "Can't allocate memory for one of csv str "  "fields";  }  break;  case u:  temp = sscanf(row.fields[i], "%u", va\_arg(args, unsigned \*));  break;  }  if (temp != 1) {  error.status\_code = PARSE\_ERROR;  error.info = "Can't parse one of arguments in scan row";  }  }  va\_end(args);  if (specifiers) {  free(specifiers);  }  return error;  }  /\* Works as printf, but output is row's fields. For format restrictions check  scan\_row function documentation \*/  Error create\_row(CSVFileLine \*row, const char \*format, ...) {  COURSE\_TASK\_SPECIFIER\_TYPE \*specifiers = NULL;  size\_t count, i, length;  Error error;  va\_list args;  char \*str;  int temp;  va\_start(args, format);  error = format\_str\_to\_enum\_list(format, (int \*\*)&specifiers, &count);  if (error.status\_code == OK) {  row->fields = malloc(sizeof(char \*) \* count);  if (!row->fields) {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info = "Can't allocate memory for row";  }  }  for (i = 0; i < count && error.status\_code == OK; i++) {  /\* Calculation of field's length to guarantely contain value \*/  switch (specifiers[i]) {  case c:  length = 1;  break;  case s:  str = va\_arg(args, char \*);  length = strlen(str) + 1;  break;  case lf:  length = DBL\_MAX\_10\_EXP + 2;  break;  case f:  length = FLT\_MAX\_10\_EXP + 2;  break;  case d:  case hd:  case hu:  case ld:  case lu:  case u:  /\* Threat other integer types as unsigned longs \*/  length = (int)ceil(log10(ULONG\_MAX) + 0.5) + 2;  break;  default:  error.status\_code = PARSE\_ERROR;  error.info = "Unknown specifier found";  length = 0;  break;  }  if (error.status\_code == OK) {  row->fields[i] = malloc(sizeof(char) \* (length + 1));  if (!row->fields[i]) {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info =  "Can't allocate memory for one of csv str "  "fields";  }  }  if (error.status\_code == OK) {  switch (specifiers[i]) {  case c:  temp = sprintf(row->fields[i], "%c", va\_arg(args, int));  break;  case d:  temp = sprintf(row->fields[i], "%d", va\_arg(args, int));  break;  case f:  temp = sprintf(row->fields[i], "%f", va\_arg(args, double));  break;  case hd:  temp = sprintf(row->fields[i], "%hd", va\_arg(args, int));  break;  case hu:  temp = sprintf(row->fields[i], "%hu", va\_arg(args, int));  break;  case ld:  temp = sprintf(row->fields[i], "%ld", va\_arg(args, long));  break;  case lf:  temp = sprintf(row->fields[i], "%f", va\_arg(args, double));  break;  case lu:  temp = sprintf(row->fields[i], "%lu",  va\_arg(args, long unsigned));  break;  case s:  temp = 1;  strcpy(row->fields[i], str);  break;  case u:  temp =  sprintf(row->fields[i], "%u", va\_arg(args, unsigned));  break;  default:  temp = 0;  break;  }  if (temp == 0) {  error.status\_code = PARSE\_ERROR;  error.info = "Can't parse one of arguments in scan row";  }  }  }  va\_end(args);  if (error.status\_code != OK) {  for (; i > 0; i--) {  if (row->fields[i - 1] != NULL) {  free(row->fields[i - 1]);  }  }  free(row->fields);  row->fields = NULL;  }  if (specifiers) {  free(specifiers);  }  return error;  }  #endif  utils.h  #ifndef COURSE\_TASK\_CSV\_ROW\_CONVERTERS\_UTILS  #define COURSE\_TASK\_CSV\_ROW\_CONVERTERS\_UTILS  #define COURSE\_TASK\_SPECIFIER\_TYPE \  enum Specifier { c = 0, d, f, hd, hu, ld, lf, lu, s, u }  #include <stdlib.h>  #include "../enums.h"  #include "../structs.h"  int specifier\_to\_enum(char specifier[3]);  /\* Converts format string to specifier enum array  output is specifier array  count is count of specifiers \*/  /\*amogus\*/  Error format\_str\_to\_enum\_list(const char \*format, int \*\*output, size\_t \*count) {  COURSE\_TASK\_SPECIFIER\_TYPE;  char buf[3];  size\_t allocated, i, j;  int temp, \*temp\_specifiers\_ptr;  Error error = {OK, ""};  \*count = 0;  allocated = 100;  \*output = (int \*)malloc(sizeof(enum Specifier) \* 100);  if (!\*output) {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info = "Can't allocate memory for specifiers array";  }  i = 0;  j = 0;  while (format[i] && error.status\_code == OK) {  if (format[i] != '%') {  buf[j++] = format[i];  if (j > 2) {  error.status\_code = UNKNOWN;  error.info = "Wrong format";  }  }  if (error.status\_code == OK &&  (format[i] == '%' || format[i + 1] == 0)) {  if (j != 0) {  buf[j] = 0;  j = 0;  temp = specifier\_to\_enum(buf);  if (temp == -1) {  error.status\_code = UNKNOWN;  error.info = "Wrong format";  } else {  if (\*count == allocated) {  allocated \*= 2;  temp\_specifiers\_ptr = (int \*)realloc(  \*output, allocated \* sizeof(enum Specifier));  if (temp\_specifiers\_ptr) {  \*output = temp\_specifiers\_ptr;  } else {  free(\*output);  \*output = NULL;  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info =  "Can't realloc more memory for specifiers "  "array";  }  }  if (error.status\_code == OK) {  (\*output)[\*count] = temp;  (\*count)++;  }  }  }  }  i++;  }  return error;  }  size\_t specifier\_to\_size(int int\_specifier) {  COURSE\_TASK\_SPECIFIER\_TYPE specifier = (enum Specifier)int\_specifier;  size\_t size;  switch (specifier) {  case c:  size = sizeof(char);  break;  case d:  size = sizeof(int);  break;  case f:  size = sizeof(float);  break;  case s:  size = sizeof(char \*);  break;  case u:  size = sizeof(unsigned);  break;  case hd:  size = sizeof(short);  break;  case hu:  size = sizeof(unsigned short);  break;  case ld:  size = sizeof(long);  break;  case lu:  size = sizeof(unsigned long);  break;  case lf:  size = sizeof(double);  break;  }  return size;  }  int specifier\_to\_enum(char specifier[3]) {  size\_t i;  COURSE\_TASK\_SPECIFIER\_TYPE specifier\_enum = -1;  i = 0;  switch (specifier[i]) {  case 'c':  specifier\_enum = c;  break;  case 'd':  specifier\_enum = d;  break;  case 'e':  case 'f':  case 'g':  specifier\_enum = f;  break;  case 's':  specifier\_enum = s;  break;  case 'u':  specifier\_enum = u;  break;  case 'h':  i++;  switch (specifier[i]) {  case 'd':  specifier\_enum = hd;  break;  case 'u':  specifier\_enum = hu;  break;  }  break;  case 'l':  i++;  switch (specifier[i]) {  case 'd':  specifier\_enum = ld;  break;  case 'u':  specifier\_enum = lu;  break;  case 'e':  case 'f':  case 'g':  specifier\_enum = lf;  break;  }  break;  }  return specifier\_enum;  }  #endif  add\_funcs.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS\_ADD\_FUNCS  #define COURSE\_TASK\_MENU\_FUNCTIONS\_ADD\_FUNCS  #include <stdio.h>  #include <stdlib.h>  #include "../enums.h"  #include "../printers.h"  #include "../structs.h"  #include "../utils.h"  #include "input\_funcs.h"  #include "utils.h"  /\* Adds new car to the end of cars list \*/  Error add\_car(ListHead \*cars, const ListHead engines) {  Error error = {OK, ""};  Car \*new\_car;  if (engines.count) {  new\_car = input\_car(engines);  if (new\_car) {  \*next\_car(cars->last) = new\_car;  new\_car->id = ((Engine \*)cars->last)->id + 1;  cars->last = new\_car;  cars->count++;  } else {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info = "Can't allocate memory for car";  }  } else {  puts("No engines. Can't add new car!");  fgetc(stdin);  }  return error;  }  /\* Adds engine type to the end of engines list \*/  Error add\_engine(ListHead \*engines) {  Engine \*new\_engine;  Error error = {OK, ""};  new\_engine = input\_engine();  if (new\_engine) {  \*next\_engine(engines->last) = new\_engine;  new\_engine->id = ((Engine \*)engines->last)->id + 1;  engines->last = new\_engine;  engines->count++;  } else {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info = "Can't allocate memory for engine";  }  return error;  }  #endif  edit\_funcs.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS\_EDIT\_FUNCS  #define COURSE\_TASK\_MENU\_FUNCTIONS\_EDIT\_FUNCS  #include <stdio.h>  #include <stdlib.h>  #include "../enums.h"  #include "../printers.h"  #include "../structs.h"  #include "../utils.h"  #include "utils.h"  /\* Returns pointer to element with id entered by user \*/  void \*select\_element(const ListHead list, unsigned \*(\*id\_getter)(void \*),  void \*\*(next\_getter)(void \*),  void (\*printer)(const void \*)) {  char confirm;  void \*element = NULL;  unsigned id;  do {  fputs("Enter id: ", stdout);  scanf("%u", &id);  fgetc(stdin);  element = find\_element\_with\_id(list, id, id\_getter, next\_getter);  if (element) {  puts("You are going to edit next element:");  printer(element);  puts("Continue (y/n)?");  confirm = fgetc(stdin);  flush\_stdin();  if (confirm == 'n') {  element = NULL;  }  } else {  printf("Element with id=%u not found. Please retry or cancel.", id);  puts("Retry (y/n)?");  confirm = fgetc(stdin);  flush\_stdin();  if (confirm == 'y') {  confirm = 0;  }  }  } while (confirm != 'y' && confirm != 'n');  return element;  }  #endif  exit\_funcs.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS\_EXIT\_FUNCS  #define COURSE\_TASK\_MENU\_FUNCTIONS\_EXIT\_FUNCS  #include <stdio.h>  #include "../csv\_file.h"  #include "../enums.h"  #include "../structs.h"  /\* Saves list to file. Uses element\_to\_row convertation function to create  row from list element, next\_getter to go through list. element\_fields\_count is  used just to set CSVFile's fields\_count parameter \*/  Error save\_list\_to\_file(ListHead list, char \*output\_filename,  size\_t element\_fields\_count,  Error (\*element\_to\_row)(void \*element,  CSVFileLine \*row),  void \*\*(\*next\_getter)(void \*element)) {  Error error = {OK, ""};  void \*element;  CSVFile csv = {0, 0, NULL};  FILE \*output;  size\_t i;  csv.fields\_count = element\_fields\_count;  if (error.status\_code == OK) {  csv.lines = (CSVFileLine \*)malloc(sizeof(CSVFileLine) \* list.count);  if (csv.lines) {  csv.lines\_count = list.count;  } else {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info = "Can't allocate memory for csv output struct";  csv.lines\_count = 0;  }  }  element = list.first;  i = 0;  while (element && error.status\_code == OK) {  error = element\_to\_row(element, csv.lines + i++);  if (error.status\_code != OK) {  csv.lines\_count = i - 1;  }  element = \*next\_getter(element);  }  if (error.status\_code == OK) {  output = fopen(output\_filename, "wb");  if (output) {  CSVFileWrite(output, csv);  fclose(output);  } else {  error.status\_code = FILE\_OPEN\_ERROR;  error.info = "Can't open output file for write";  }  }  CSVFileFree(&csv);  return error;  }  #endif  input\_funcs.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS\_INPUT\_FUNCS  #define COURSE\_TASK\_MENU\_FUNCTIONS\_INPUT\_FUNCS  #include <stdio.h>  #include <stdlib.h>  #include "../enums.h"  #include "../getters.h"  #include "../printers.h"  #include "../structs.h"  #include "../utils.h"  #include "utils.h"  #define MAX\_STR\_LEN 512  /\* Allocating and reading from stdin new car \*/  Car \*input\_car(const ListHead engines) {  Car \*car;  unsigned id;  if (engines.count) {  car = (Car \*)malloc(sizeof(Car));  if (car) {  car->brand = (char \*)malloc(sizeof(char) \* MAX\_STR\_LEN);  car->model = (char \*)malloc(sizeof(char) \* MAX\_STR\_LEN);  }  if (!car || !car->brand || !car->model) {  free\_car((void \*\*)&car);  } else {  fputs("Enter car cost and year of issue: ", stdout);  scanf("%f %d", &car->cost, &car->year\_of\_issue);  fgetc(stdin);  fputs("Enter car brand: ", stdout);  fgets(car->brand, 21, stdin);  remove\_last\_symbol(car->brand);  fputs("Enter car model: ", stdout);  fgets(car->model, MAX\_STR\_LEN, stdin);  remove\_last\_symbol(car->model);  do {  print\_engines\_list(engines);  fputs("Enter car's engine id: ", stdout);  scanf("%u", &id);  fgetc(stdin);  car->engine = (Engine \*)find\_element\_with\_id(  engines, id, engine\_get\_id, next\_engine);  if (!car->engine) {  printf(  "Engine with id=%u not found. Please, "  "retry.\n",  id);  }  } while (!car->engine);  car->next = NULL;  }  }  return car;  }  /\* Allocating and reading from stdin new engine \*/  Engine \*input\_engine() {  Engine \*engine;  engine = (Engine \*)malloc(sizeof(Engine));  if (engine) {  engine->name = (char \*)malloc(sizeof(char) \* MAX\_STR\_LEN);  }  if (!engine || !engine->name) {  free\_engine((void \*\*)&engine);  } else {  fputs("Enter engine name (up to 20 characters): ", stdout);  fgets(engine->name, 21, stdin);  remove\_last\_symbol(engine->name);  engine->next = NULL;  fputs("Enter engine volume and power: ", stdout);  scanf("%f %d", &engine->volume, &engine->power);  }  return engine;  }  #endif  search\_funcs.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS\_SEARCH\_FUNCS  #define COURSE\_TASK\_MENU\_FUNCTIONS\_SEARCH\_FUNCS  #include <stddef.h>  #include <string.h>  #include "../structs.h"  /\* Non-deep search in list by field of dynamic size with additional string  search feature \*/  char search(const ListHead list, void \*field\_value, size\_t field\_size,  void \*(\*field\_getter)(void \*), void \*\*(\*next\_getter)(void \*),  void (\*printer)(const void \*), char is\_str) {  void \*iter = list.first;  char found = 0;  while (iter) {  if (is\_str) {  if (strstr(\*((char \*\*)field\_getter(iter)), (char \*)field\_value) !=  NULL) {  printer(iter);  found = 1;  }  } else {  if (memcmp(field\_value, field\_getter(iter), field\_size) == 0) {  printer(iter);  found = 1;  }  }  iter = \*next\_getter(iter);  }  return found;  }  #endif  utils.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS\_UTILS  #define COURSE\_TASK\_MENU\_FUNCTIONS\_UTILS  #include <stddef.h>  #include "../getters.h"  #include "../structs.h"  /\* Finds element which \*id\_getter(element) equals to id parameter \*/  void \*find\_element\_with\_id(ListHead list, unsigned id,  unsigned \*(\*id\_getter)(void \*),  void \*\*(\*next\_getter)(void \*)) {  void \*element = list.first;  char found = 0;  while (element && !found) {  if (\*id\_getter(element) == id) {  found = 1;  } else {  element = \*next\_getter(element);  }  }  return element;  }  /\* Universal element deletion \*/  void delete\_list\_element(ListHead \*list, void \*element,  void \*\*(\*next\_getter)(void \*),  void (\*free\_func)(void \*\*)) {  void \*iter, \*prev;  iter = list->first;  prev = NULL;  while (iter) {  if (iter == element) {  if (iter == list->first) {  list->first = \*next\_getter(iter);  }  if (iter == list->last) {  list->last = prev;  }  if (prev) {  \*next\_getter(prev) = \*next\_getter(iter);  }  free\_func(&iter);  list->count--;  } else {  prev = iter;  iter = \*next\_getter(iter);  }  }  }  #endif  sort\_funcs.h  #ifndef COURSE\_TASK\_MENU\_FUNCTIONS\_SORT\_FUNCS  #define COURSE\_TASK\_MENU\_FUNCTIONS\_SORT\_FUNCS  #include <stdlib.h>  #include <string.h>  #include "../enums.h"  #include "../getters.h"  #include "../structs.h"  /\* Compares two elements based on their type. Returns -1 if value1 < value2,  0 if value1 == value2 and 1 if value1 > value2 \*/  int compare(void \*value1, void \*value2, VariableType type);  /\* List universal sorter. Returns list of ListElement elements.  That elements' data fields are pointers to to\_sort array elements \*/  Error sort\_list(ListHead to\_sort, ListHead \*output,  void \*(\*field\_getter)(void \*), void \*\*(\*next\_getter)(void \*),  VariableType type) {  Error error = {OK, ""};  void \*element = NULL, \*element2 = NULL;  void \*temp = NULL;  output->count = 0;  output->first = NULL;  output->last = NULL;  /\* Copy elements to output array \*/  element = to\_sort.first;  while (element && error.status\_code == OK) {  if (output->last) {  \*next\_list\_element(output->last) = malloc(sizeof(ListElement));  output->last = \*next\_list\_element(output->last);  } else {  output->last = malloc(sizeof(ListElement));  output->first = output->last;  }  if (output->last) {  \*list\_element\_get\_data(output->last) = element;  output->count++;  } else {  error.status\_code = MEMORY\_ALLOCATION\_ERROR;  error.info = "Can't allocate memory for new list element";  }  element = \*next\_getter(element);  }  \*next\_list\_element(output->last) = NULL;  /\* Sort elements by selected field (bubble sort) \*/  if (error.status\_code == OK) {  element = output->first;  while (element) {  element2 = \*next\_list\_element(element);  while (element2) {  if (compare(field\_getter(\*list\_element\_get\_data(element)),  field\_getter(\*list\_element\_get\_data(element2)),  type) > 0) {  temp = \*list\_element\_get\_data(element);  \*list\_element\_get\_data(element) =  \*list\_element\_get\_data(element2);  \*list\_element\_get\_data(element2) = temp;  }  element2 = \*next\_list\_element(element2);  }  element = \*next\_list\_element(element);  }  } else {  free\_list(output, free\_list\_element, next\_list\_element);  }  return error;  }  int compare(void \*value1, void \*value2, VariableType type) {  int result;  switch (type) {  case CHAR:  result = \*((char \*)value1) > \*((char \*)value2)  ? 1  : (\*((char \*)value1) < \*((char \*)value2) ? -1 : 0);  break;  case DOUBLE:  result = \*((double \*)value1) > \*((double \*)value2)  ? 1  : (\*((double \*)value1) < \*((double \*)value2) ? -1 : 0);  break;  case FLOAT:  result = \*((float \*)value1) > \*((float \*)value2)  ? 1  : (\*((float \*)value1) < \*((float \*)value2) ? -1 : 0);  break;  case INT:  result = \*((int \*)value1) > \*((int \*)value2)  ? 1  : (\*((int \*)value1) < \*((int \*)value2) ? -1 : 0);  break;  case STRING:  result = strcmp(\*((char \*\*)value1), \*((char \*\*)value2));  break;  case UNSIGNED:  result =  \*((unsigned \*)value1) > \*((unsigned \*)value2)  ? 1  : (\*((unsigned \*)value1) < \*((unsigned \*)value2) ? -1 : 0);  break;  }  return result;  }  #endif |

**Пример выполнения программы.**

**Заключение.**

Используемые заголовочные файлы стандартной библиотеки:

1. stdio.h
   1. fputs – записывает строку в файл
   2. fputc – записывает символ в файл
   3. scanf – считывает данные
   4. fgetc – считывает символ из файла
   5. fopen – открывает файл
   6. fseek – ищет в файле
   7. printf – выводит аргументы
   8. puts – выводит строку
   9. fwrite – записывает объекты в файл
   10. fgets – считывает символы из файла
   11. sscanf – считывает данные из массива
2. stdlib.h
   1. system – передает команду командному процессору
   2. free – освобождает память
   3. malloc – возвращает адрес на первый байт области памяти
   4. realloc – изменяет величину выделенной памяти на новую
3. string.h
   1. strstr – ищет по строке
   2. memcmp – сравнивает символы
   3. strcmp – сравнивает строки
   4. strcpy – копирует из одной строки в другую
   5. strlen – считает длину строки
4. stdarg.h
   1. va\_arg – осуществляет передачу функции переменного числа параметров
   2. va\_list – определяет тип va\_list
   3. va\_start – осуществляет передачу функции переменного числа параметров
5. stddef.h
   1. size\_t – определяет тип size\_t
6. float.h
   1. DBL\_MAX\_10\_EXP – максимальное целое число, при котором число 10, возведенное в степень этого числа, является представимым числом с плавающей запятой.
   2. FLT\_MAX\_10\_EXP – максимальное целое число, при котором число 10, возведенное в степень этого числа, является представимым числом с плавающей запятой.
7. limits.h
   1. ULONG\_MAX – максимальное значение для беззнакового long int
8. math.h
   1. log10 – считает десятичный логарифм числа
   2. ceil – считает наименьшее целое, которое не меньше заданного числа

В результате выполнения работы изучена работа со структурами в языке C и получены практические навыки в создании электронных картотек.