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.

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

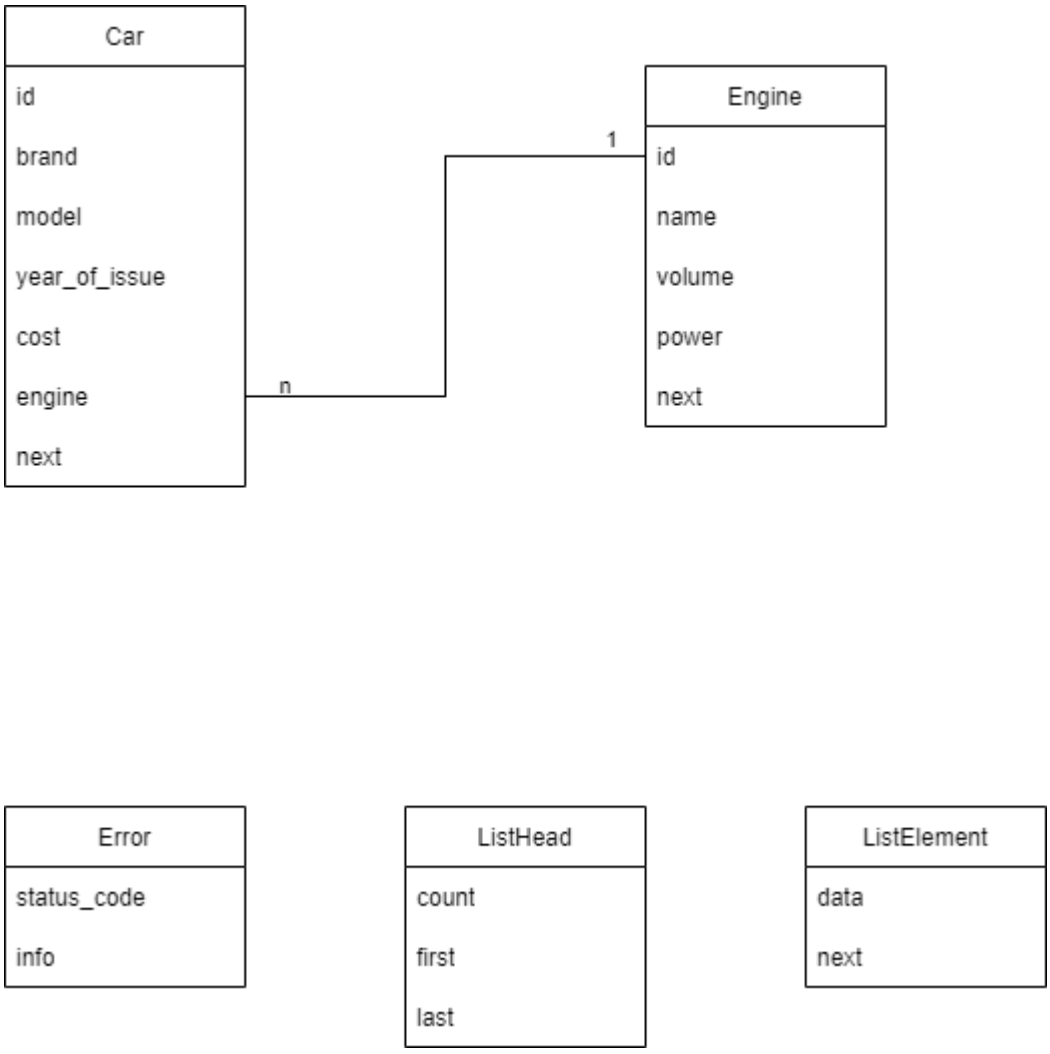
Задача работы представляет собой создание электронной картотеки автомобилей, содержащей таблицу с данными об автомобилях и таблицу с данными о двигателях

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

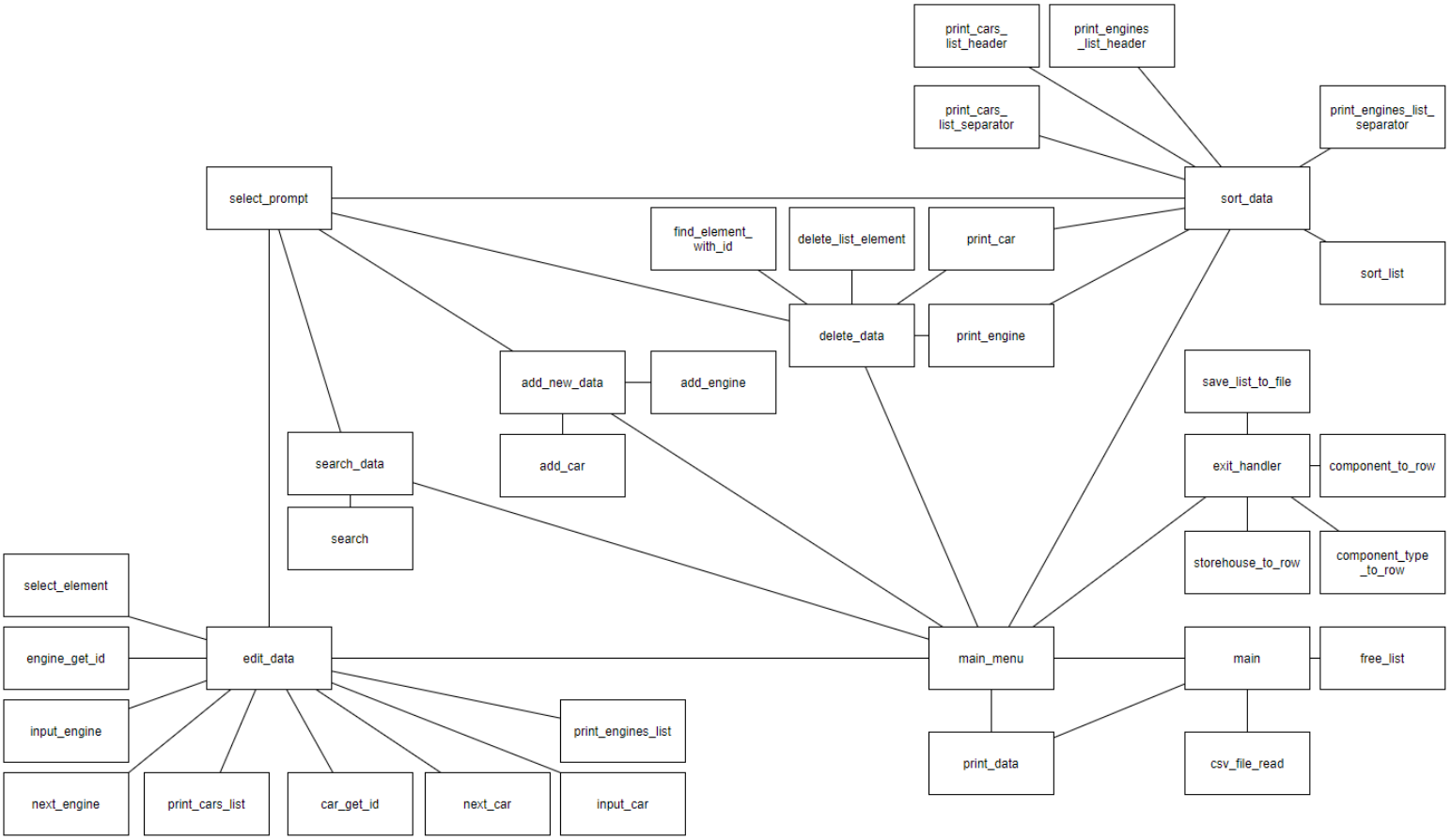
Структура Engine – содержит данные о двигателях

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

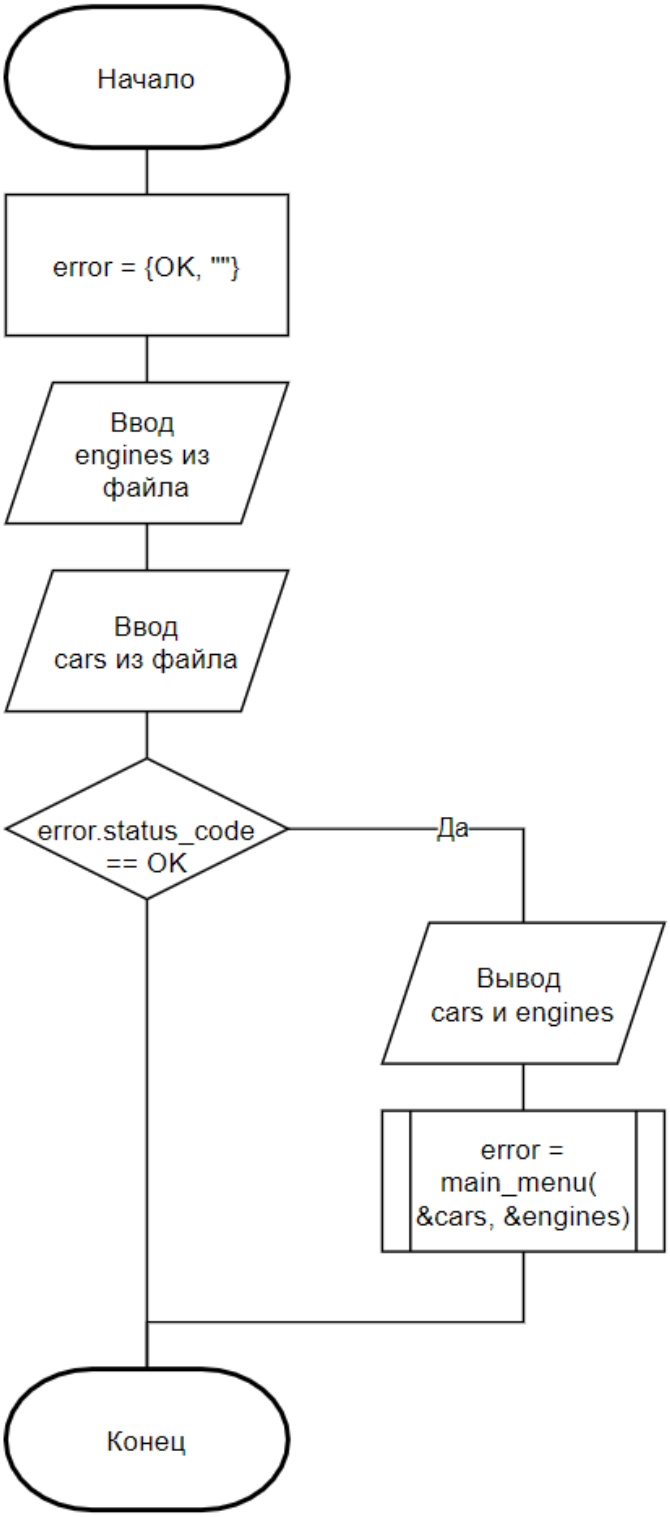
specifier_to_size

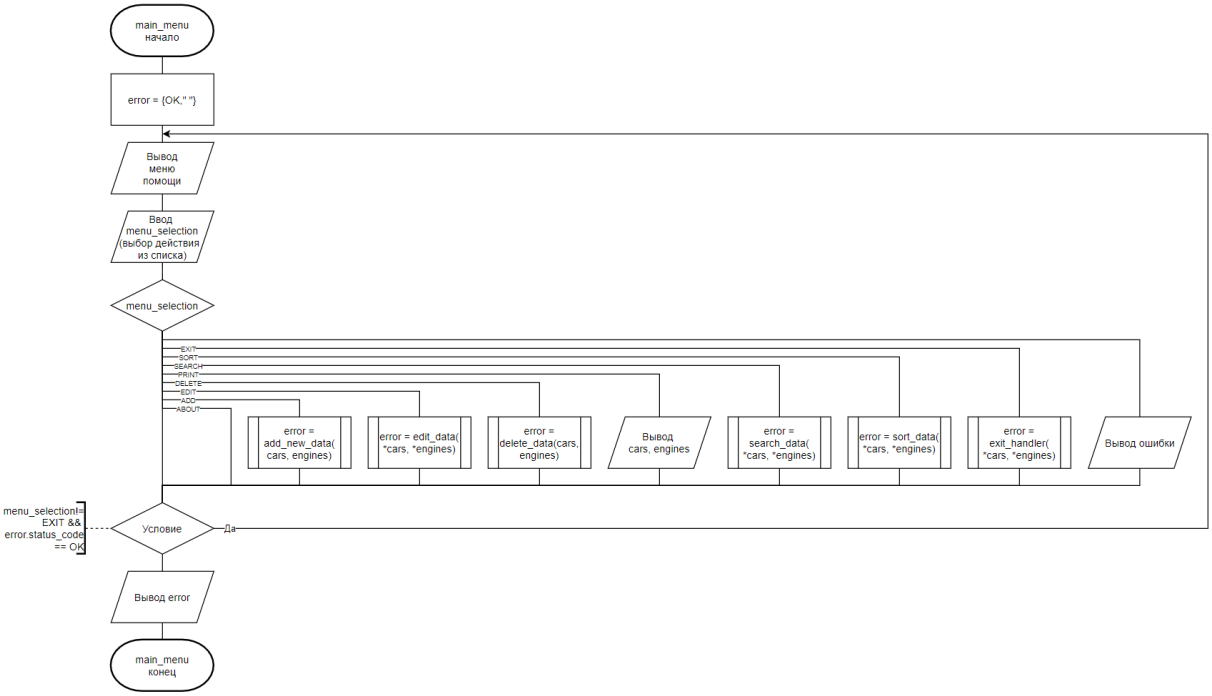
№	Имя переменной	Тип	Назначение
1	int_specifier	int	Целочисленный спецификатор
2	specifier	COURSE_TASK_ SPECIFIER_TYPE	Спецификатор
3	size	size_t	Размер

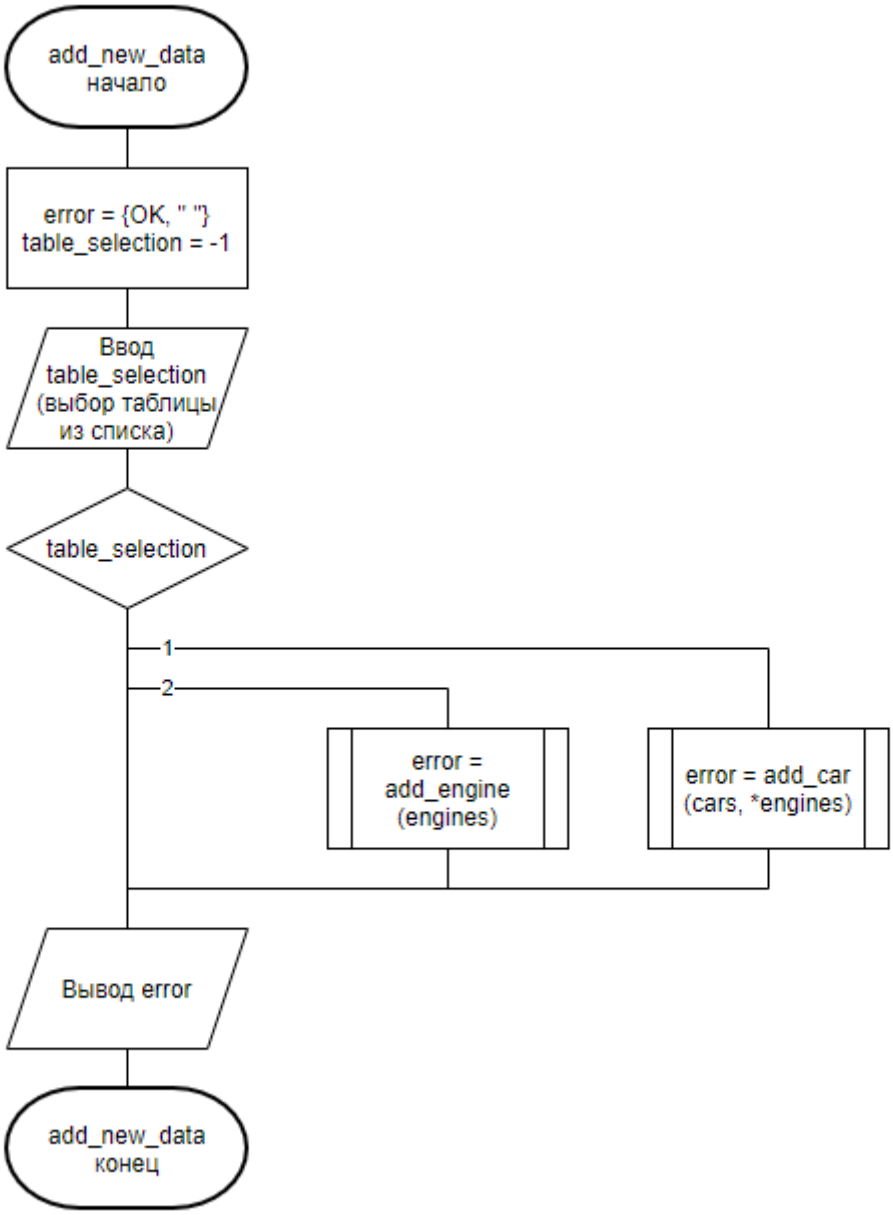
specifier_to_enum

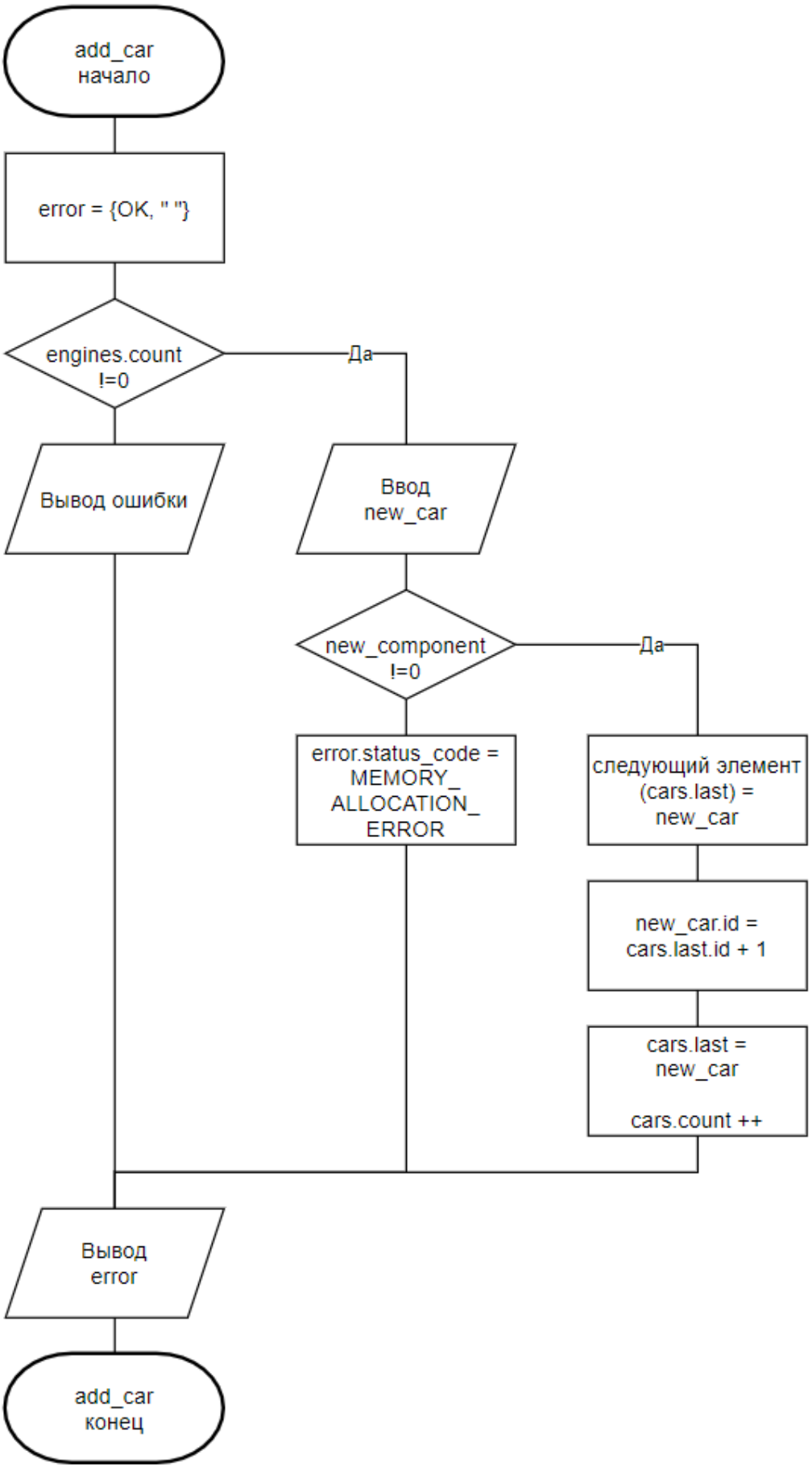
№	Имя переменной	Тип	Назначение
1	specifier_enum	COURSE_TASK_ SPECIFIER_TYPE	Спецификатор перечислений
2	specifier	char[3]	Спецификатор
3	i	size_t	Переменная для цикла for

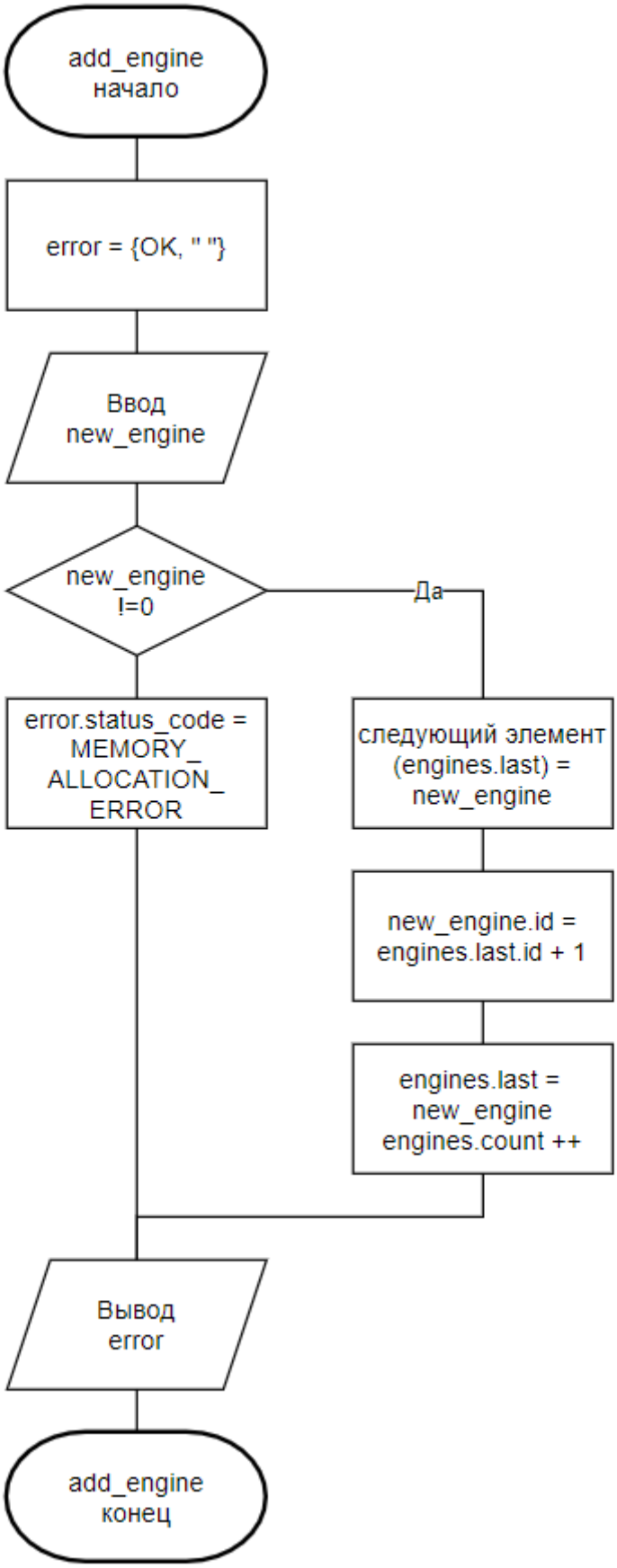
Схема алгоритма.

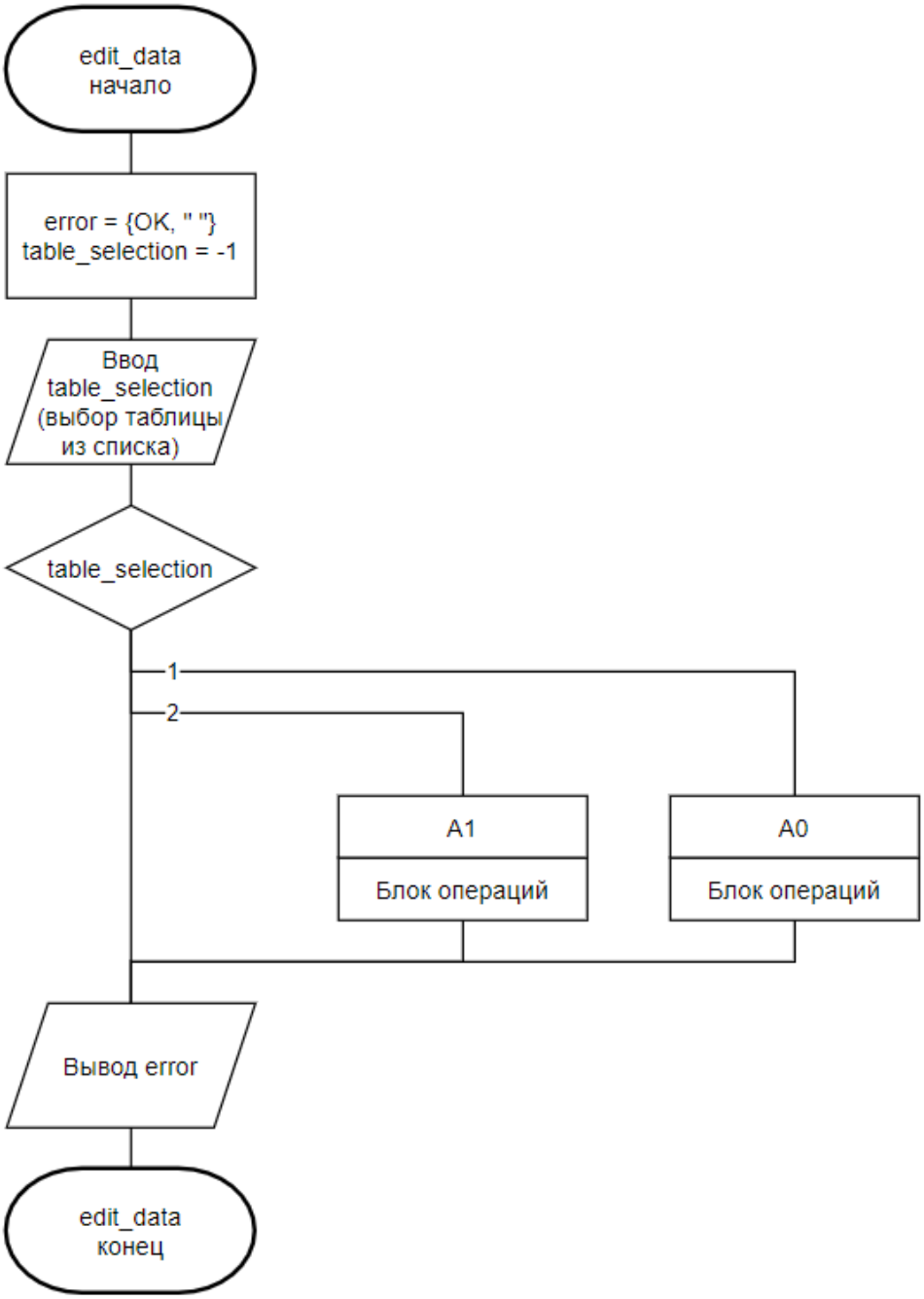


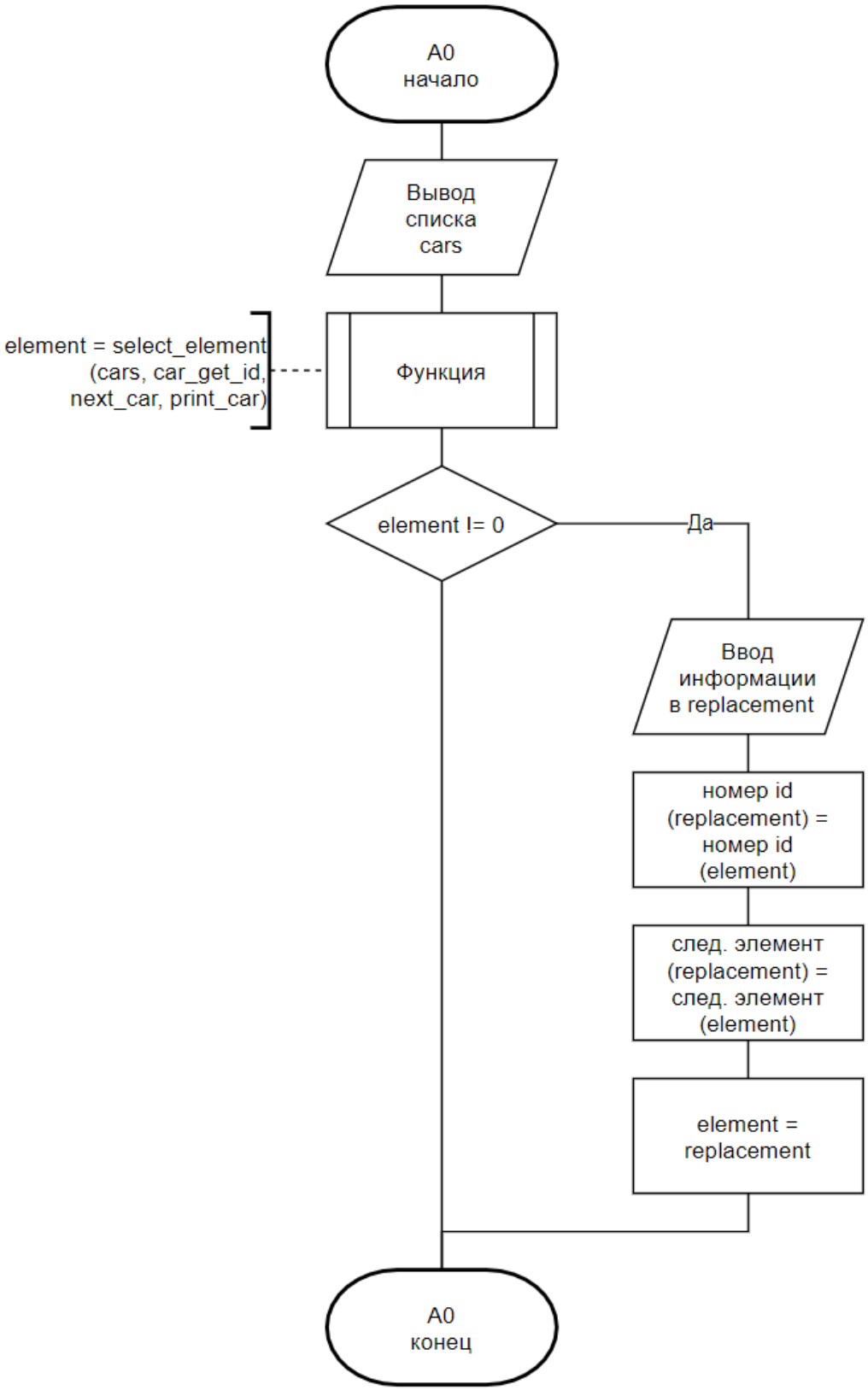

main_menu
начало
error = {OK," "}
Вывод
меню
помощи
Ввод
menu_selection
(выбор действия
из списка)
menu_selection
EXIT
SORT
SEARCH
PRINT
DELETE
EDIT
ADD
ABOUT
error =
add_new_data(
cars, engines)
error = edit_data(
*cars, *engines)
error =
delete_data(cars,
engines)
Вывод
cars, engines
error =
search_data(
*cars, *engines)
error = sort_data(
*cars, *engines)
error =
exit_handler(
*cars, *engines)
Вывод ошибки
menu_selection!=
EXIT &&
error.status_code
== OK
Условие
Да
Вывод error
main_menu
конец

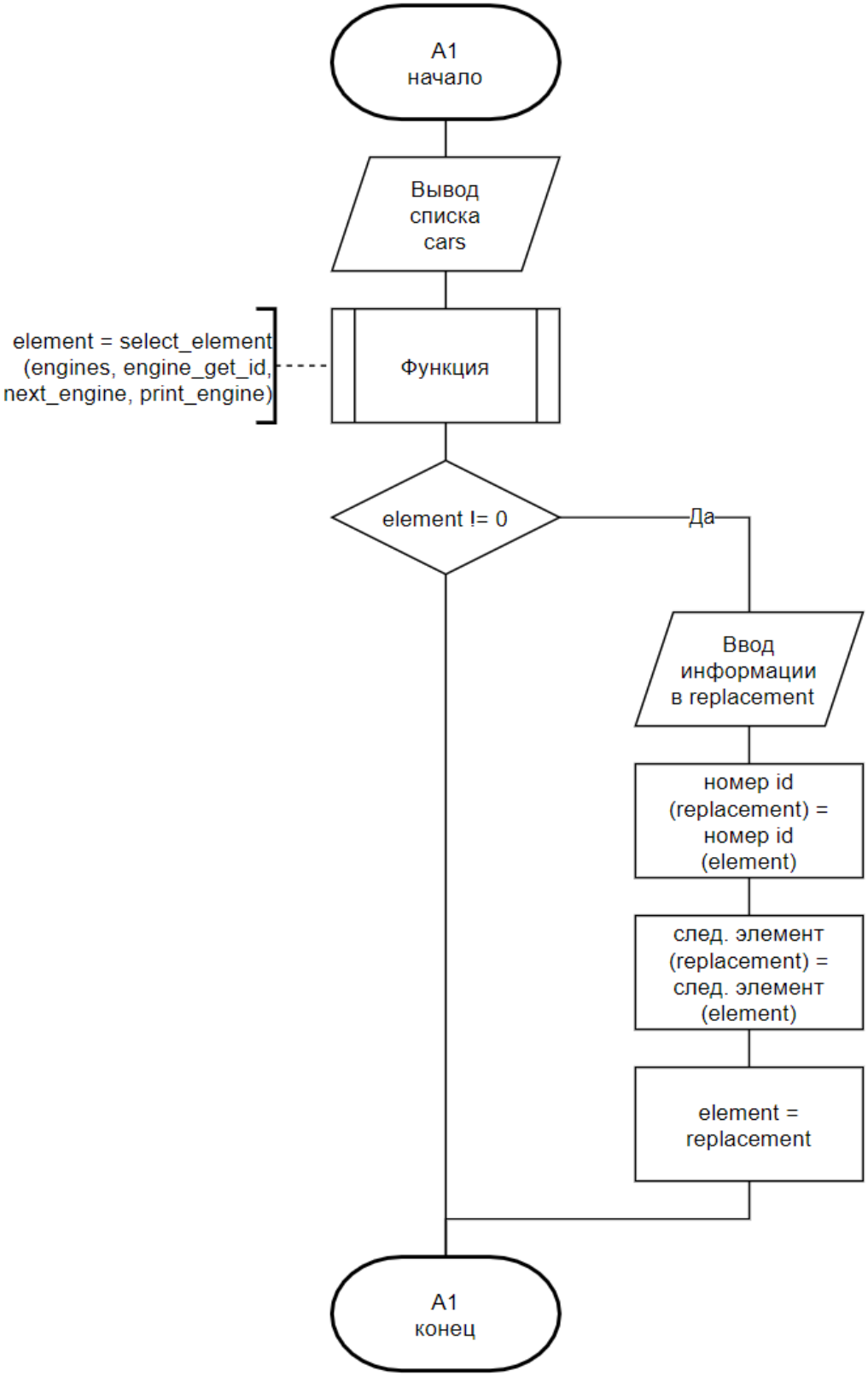

add_new_data
начало
error = {OK, '' '}
table_selection = -1
Ввод
table_selection
(выбор таблицы
из списка)
table_selection
1
2
error =
add_engine
(engines)
error = add_car
(cars, *engines)
Вывод error
add_new_data
конец

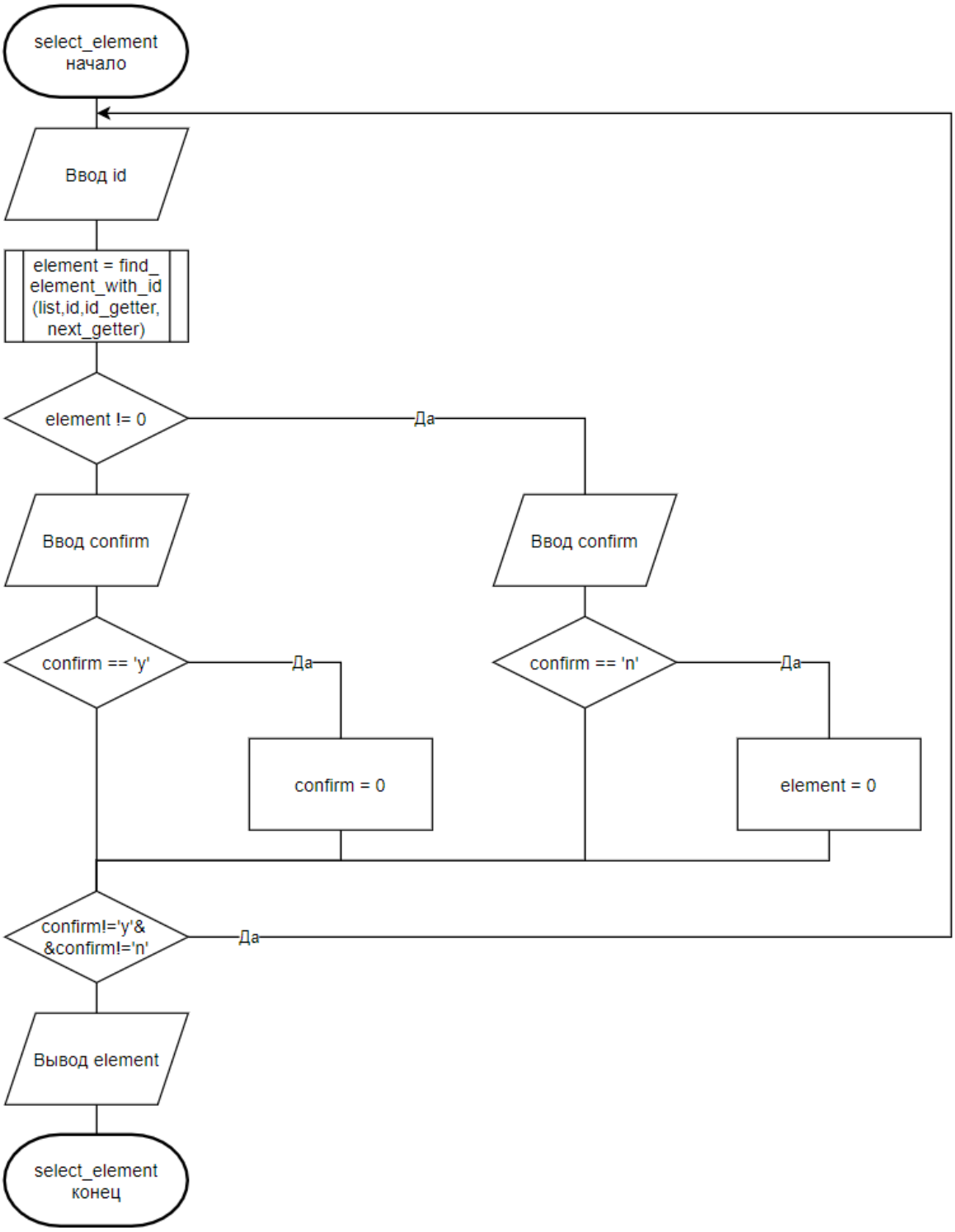

add_car
начало
error = {OK, " "}
engines.count
!=0
Да
Вывод ошибки
Ввод
new_car
new_component
!=0
Да
error.status_code =
MEMORY_
ALLOCATION_
ERROR
следующий элемент
(cars.last) =
new_car
new_car.id =
cars.last.id + 1
cars.last =
new_car
cars.count ++
Вывод
error
add_car
конец

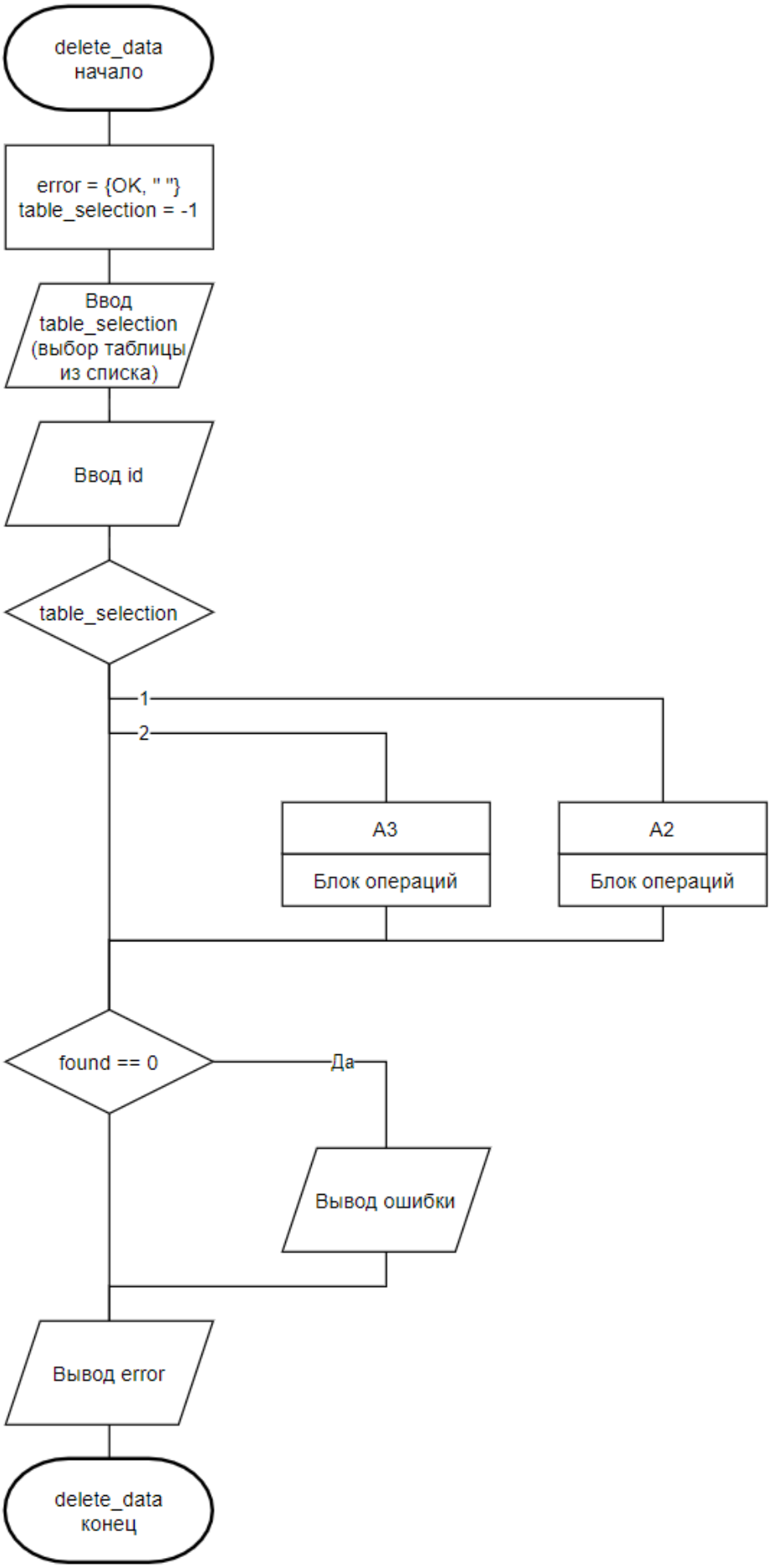

add_engine
начало
error = {OK, " "}
Ввод
new_engine
new_engine
!=0
Да
error.status_code =
MEMORY_
ALLOCATION_
ERROR
следующий элемент
(engines.last) =
new_engine
new_engine.id =
engines.last.id + 1
engines.last =
new_engine
engines.count ++
Вывод
error
add_engine
конец

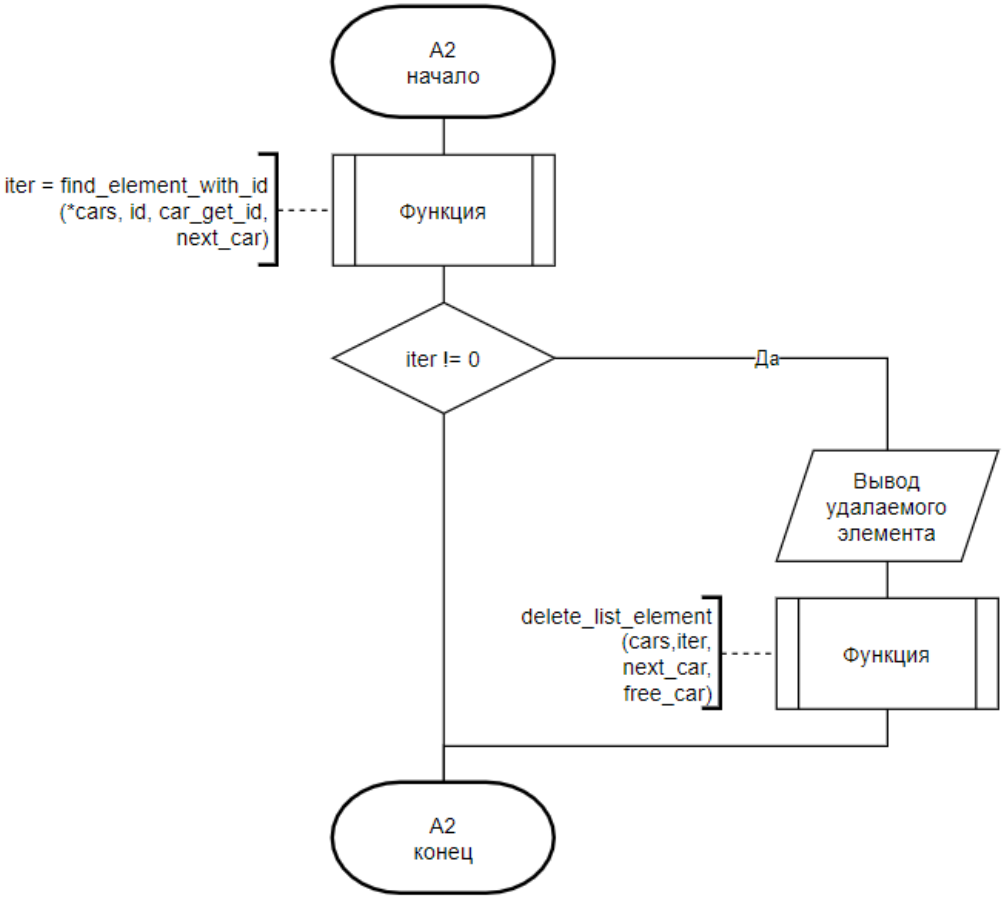

edit_data
начало
error = {OK, " "}
table_selection = -1
Ввод
table_selection
(выбор таблицы
из списка)
table_selection
1
2
A1
Блок операций
A0
Блок операций
Вывод error
edit_data
конец

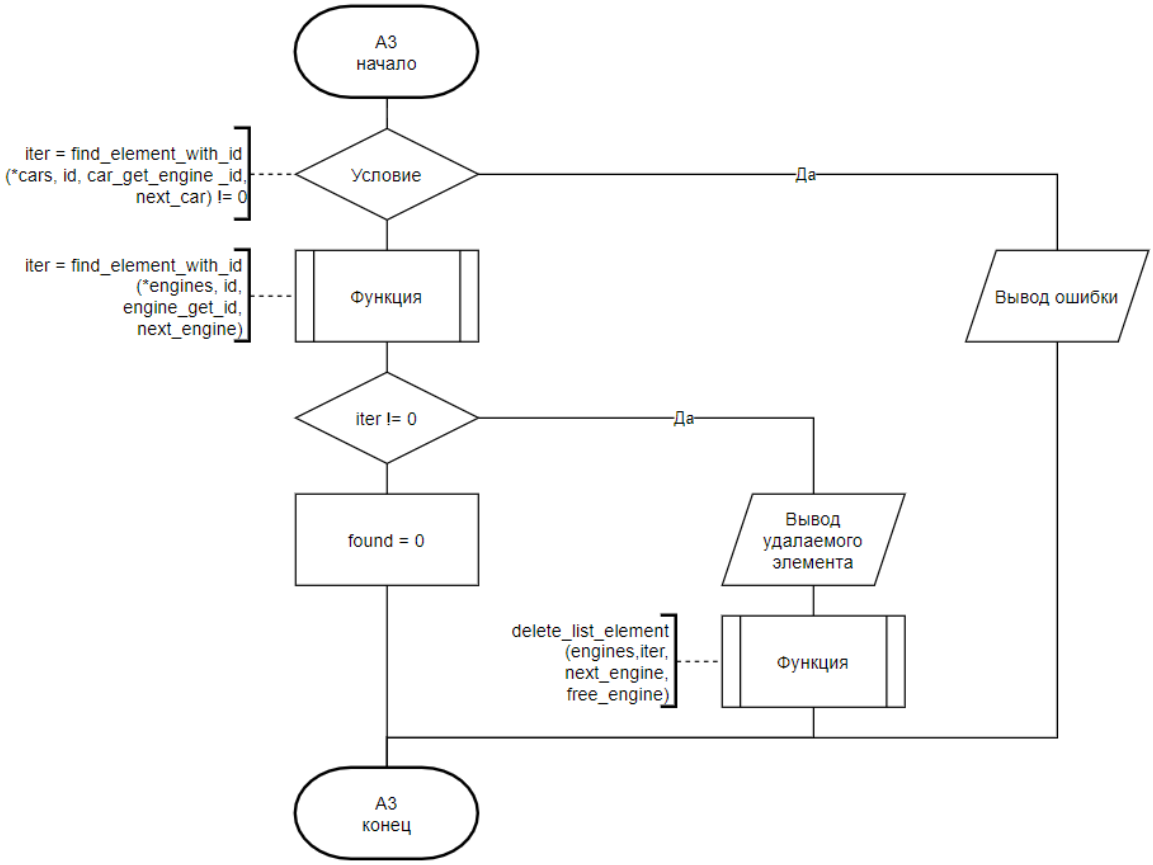

A0
начало
Вывод
списка
cars
element = select_element
(cars, car_get_id,
next_car, print_car)
Функция
element != 0
Да
Ввод
информации
в replacement
номер id
(replacement) =
номер id
(element)
след. элемент
(replacement) =
след. элемент
(element)
element =
replacement
A0
конец

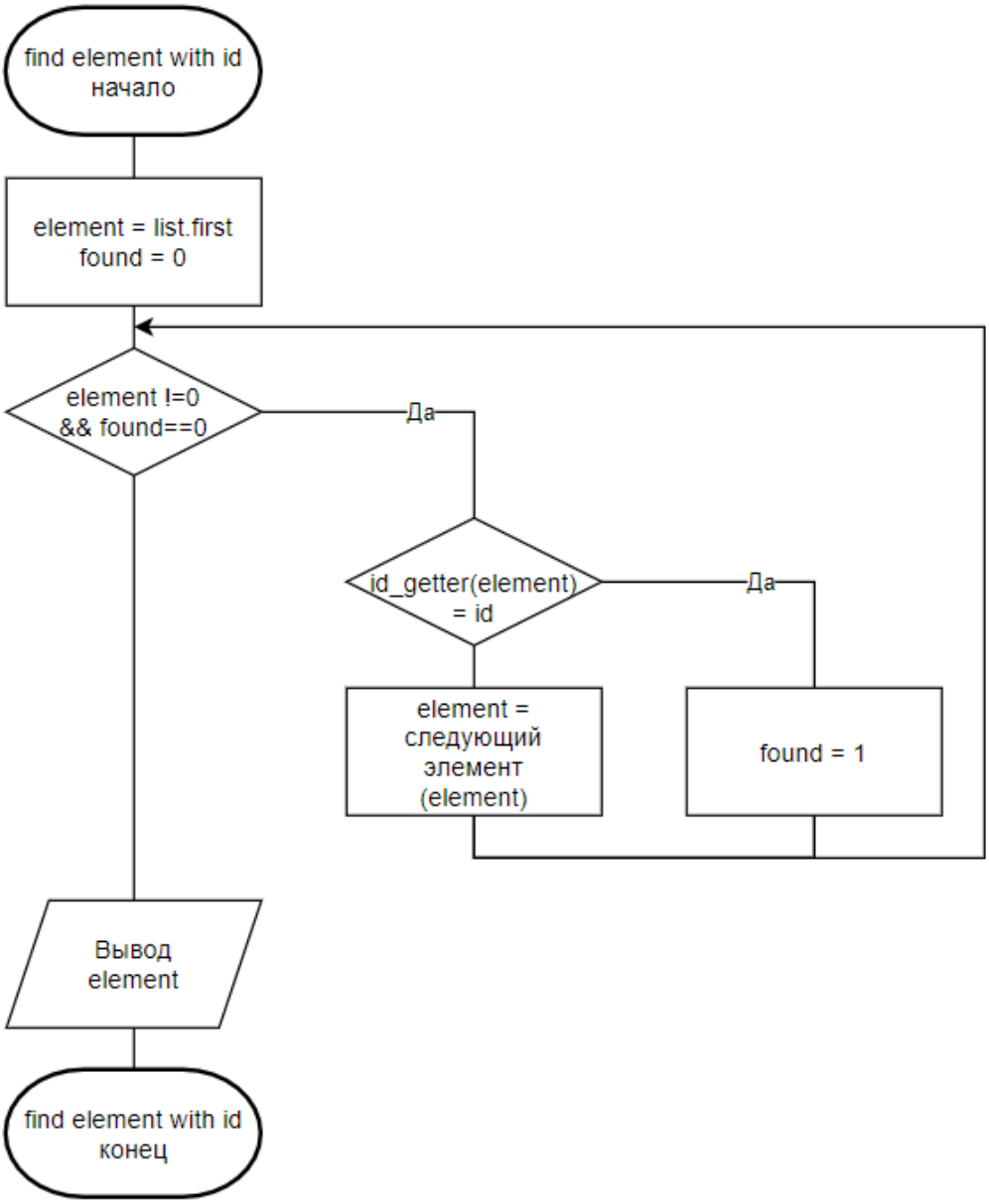

A1
начало
Вывод
списка
cars
element = select_element
(engines, engine_get_id,
next_engine, print_engine)
Функция
element != 0
Да
Ввод
информации
в replacement
номер id
(replacement) =
номер id
(element)
след. элемент
(replacement) =
след. элемент
(element)
element =
replacement
A1
конец

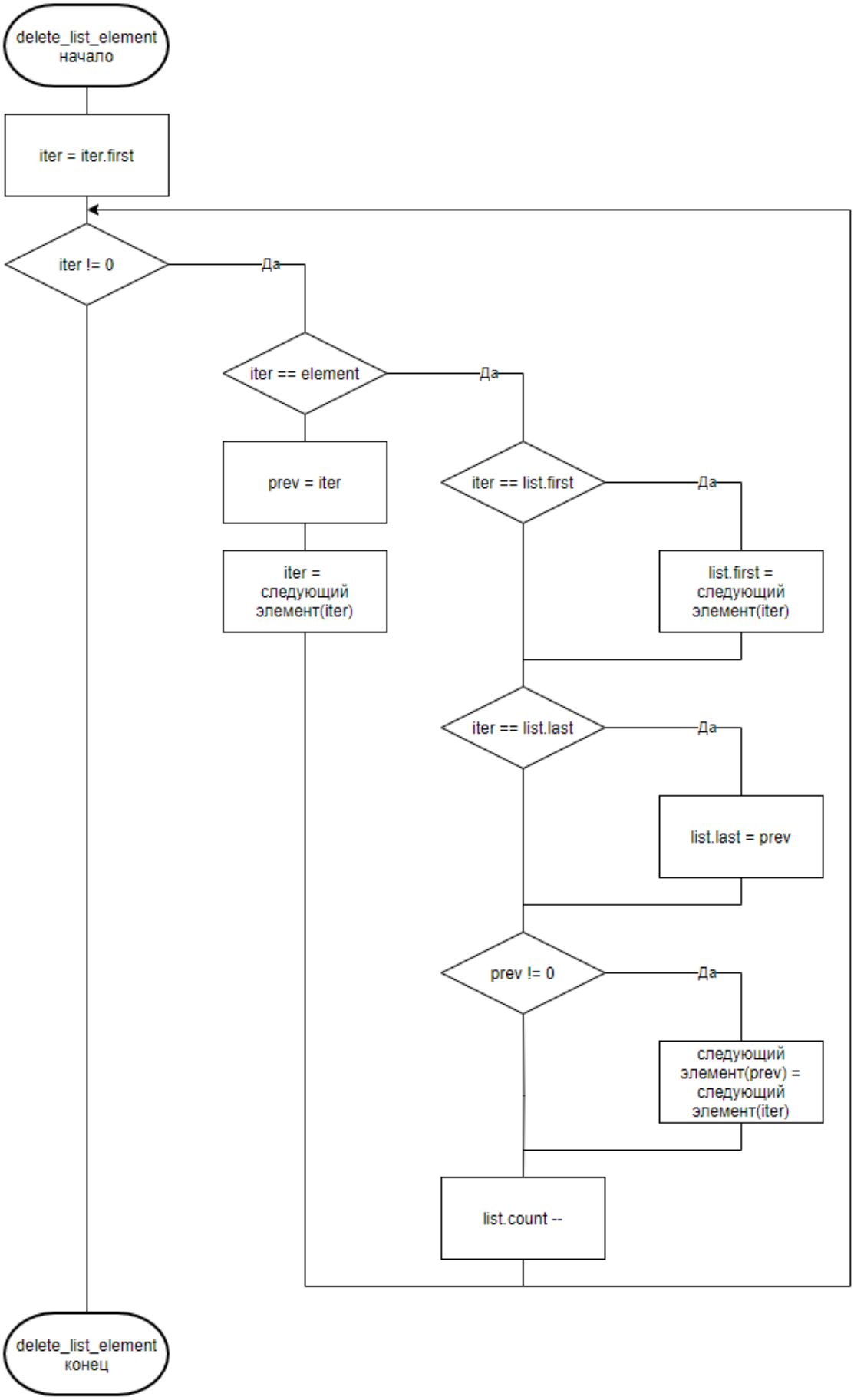

select_element
начало
Ввод id
element = find_
element_with_id
(list,id,id_getter,
next_getter)
element != 0
Да
Ввод confirm
Ввод confirm
confirm == 'y'
Да
confirm == 'n'
Да
confirm = 0
element = 0
confirm!='y'&
&confirm!='n'
Да
Вывод element
select_element
конец

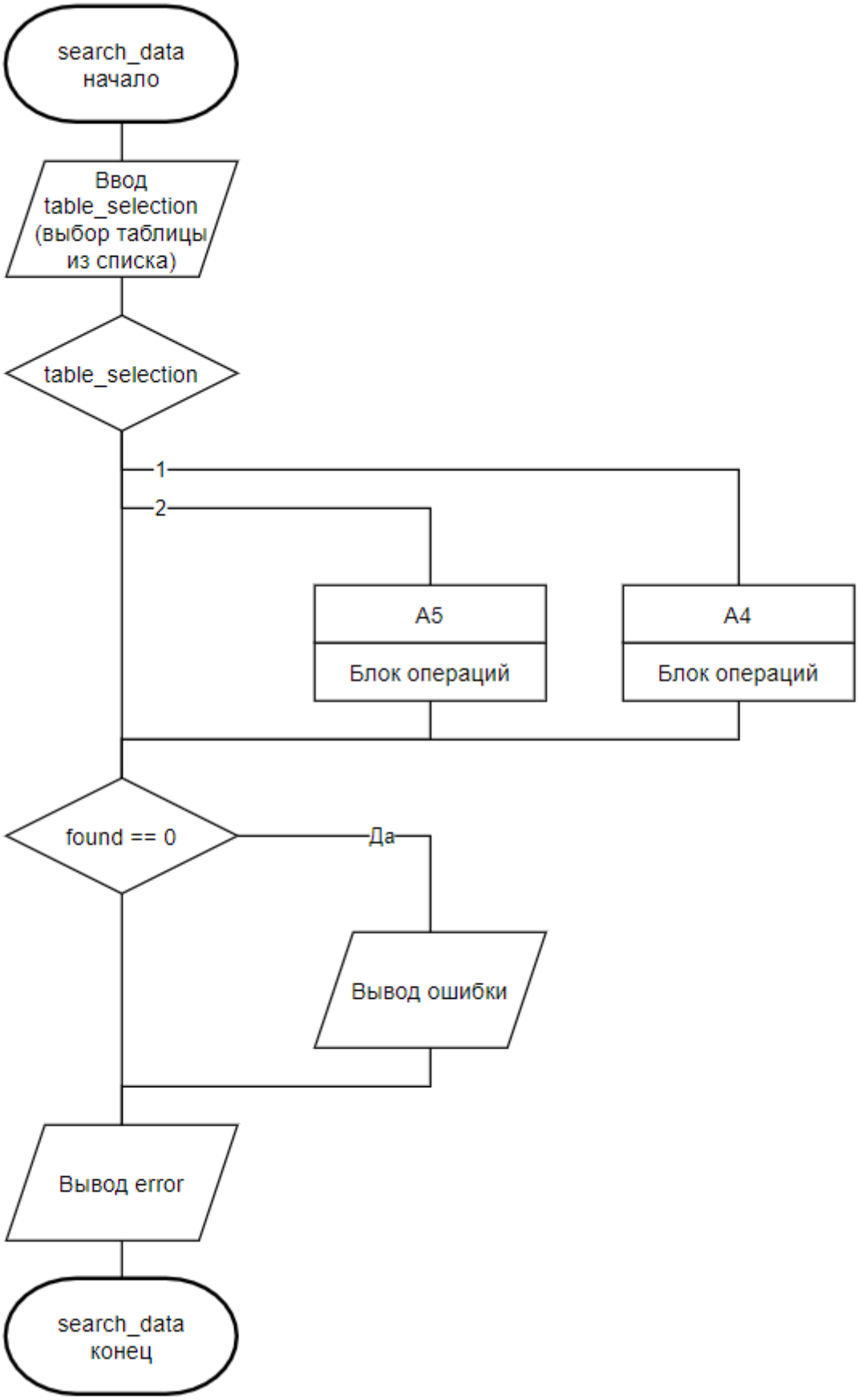

delete_data
начало
error = {OK, " "}
table_selection = -1
Ввод
table_selection
(выбор таблицы
из списка)
Ввод id
table_selection
1
2
A3
Блок операций
A2
Блок операций
found == 0
Да
Вывод ошибки
Вывод error
delete_data
конец

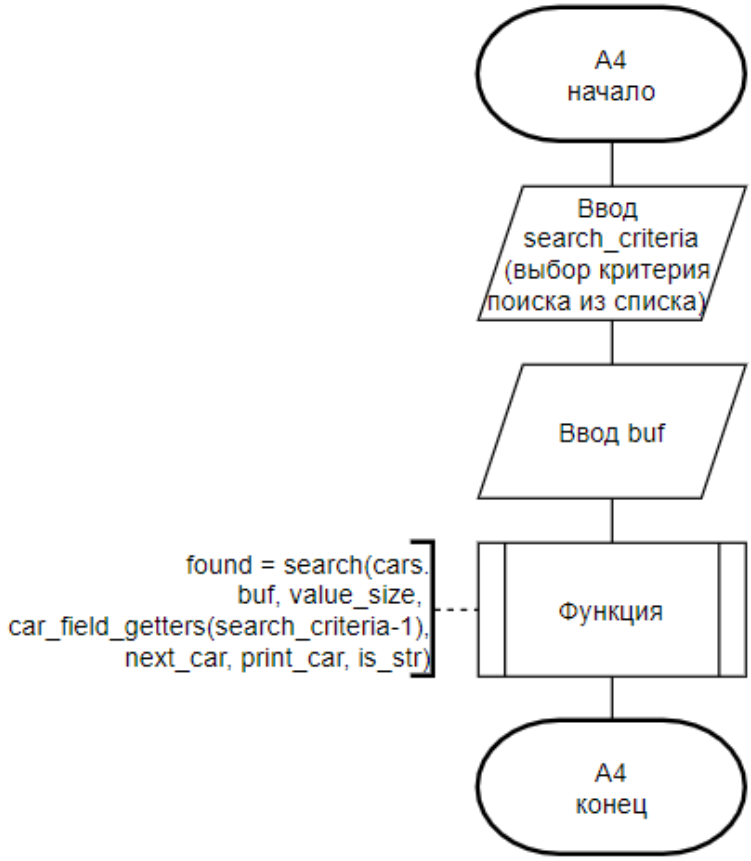

A2
начало
iter = find_element_with_id
(*cars, id, car_get_id,
next_car)
Функция
iter != 0
Да
Вывод
удалаемого
элемента
delete_list_element
(cars,iter,
next_car,
free_car)
Функция
A2
конец

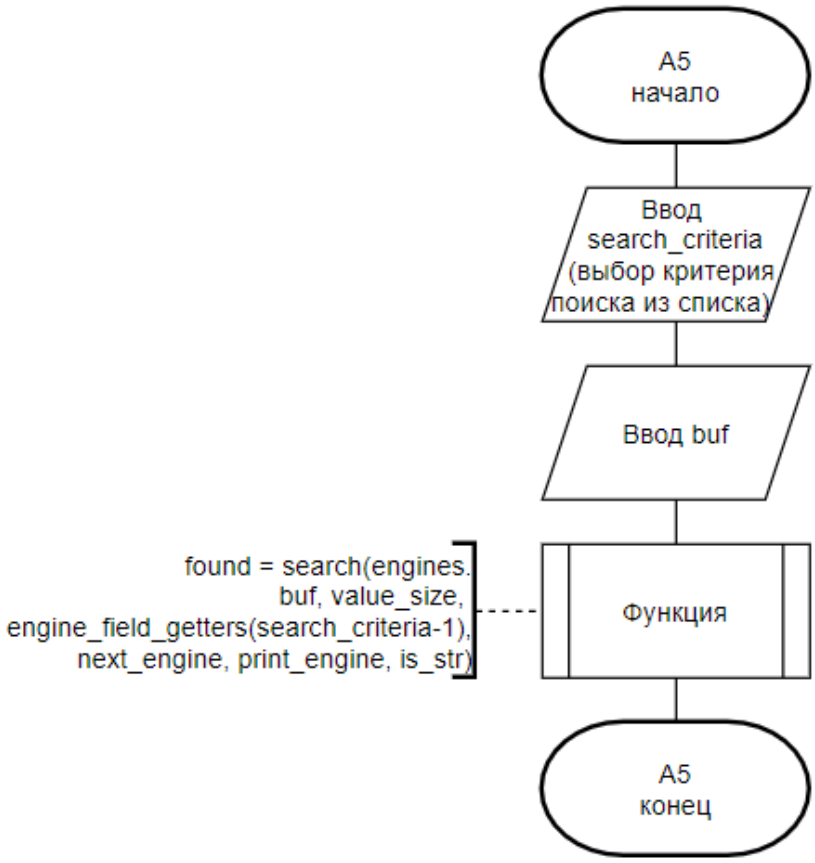

АЗ
начало
iter = find_element_with_id
(*cars, id, car_get_engine _id,
next_car) != 0
Условие
Да
iter = find_element_with_id
(*engines, id,
engine_get_id,
next_engine)
Функция
Вывод ошибки
iter != 0
Да
found = 0
Вывод
удалаемого
элемента
delete_list_element
(engines,iter,
next_engine,
free_engine)
Функция
АЗ
конец

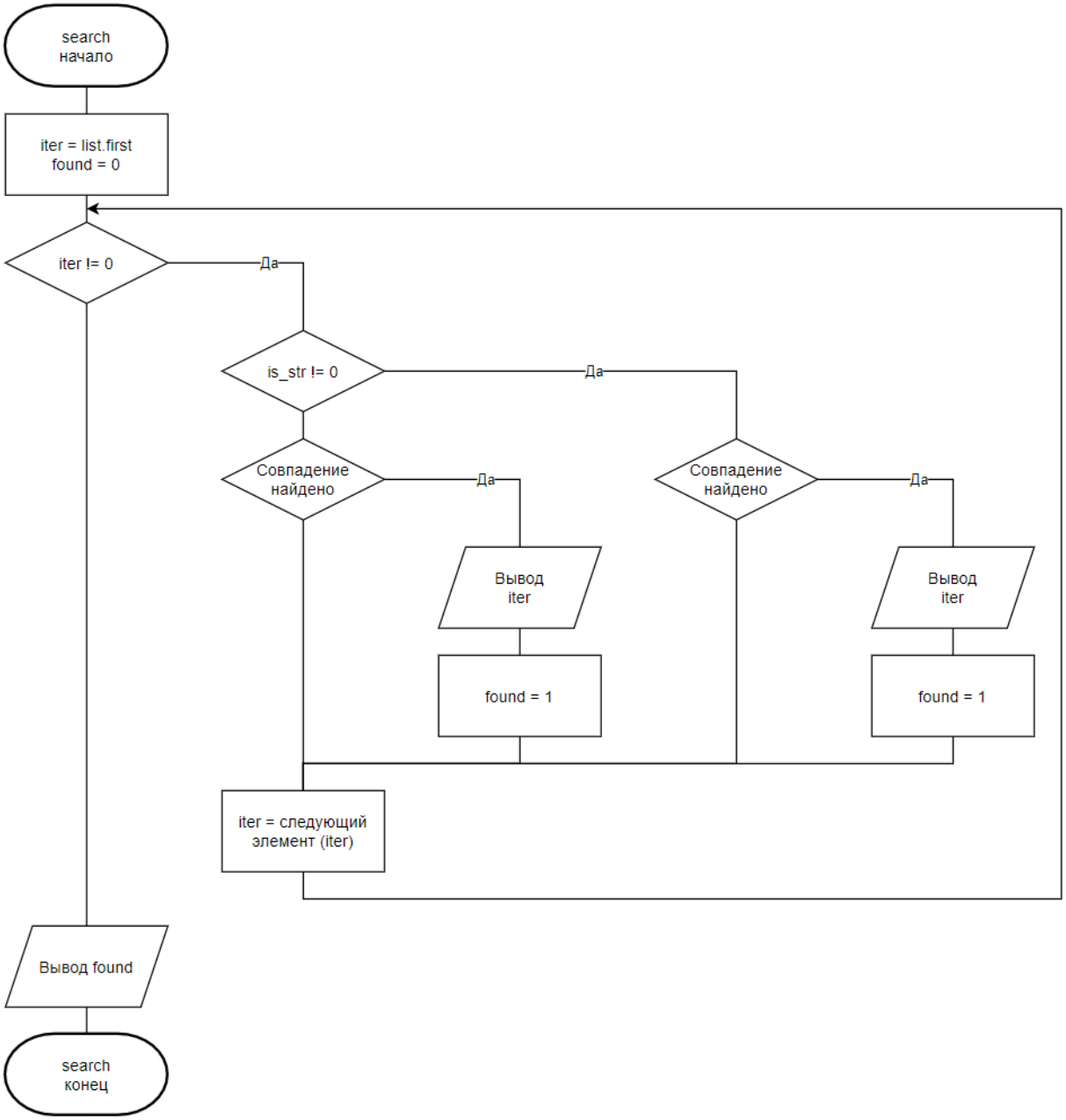

find element with id
начало
element = list.first
found = 0
element !=0
&& found==0
Да
id_getter(element)
= id
Да
element =
следующий
элемент
(element)
found = 1
Вывод
element
find element with id
конец

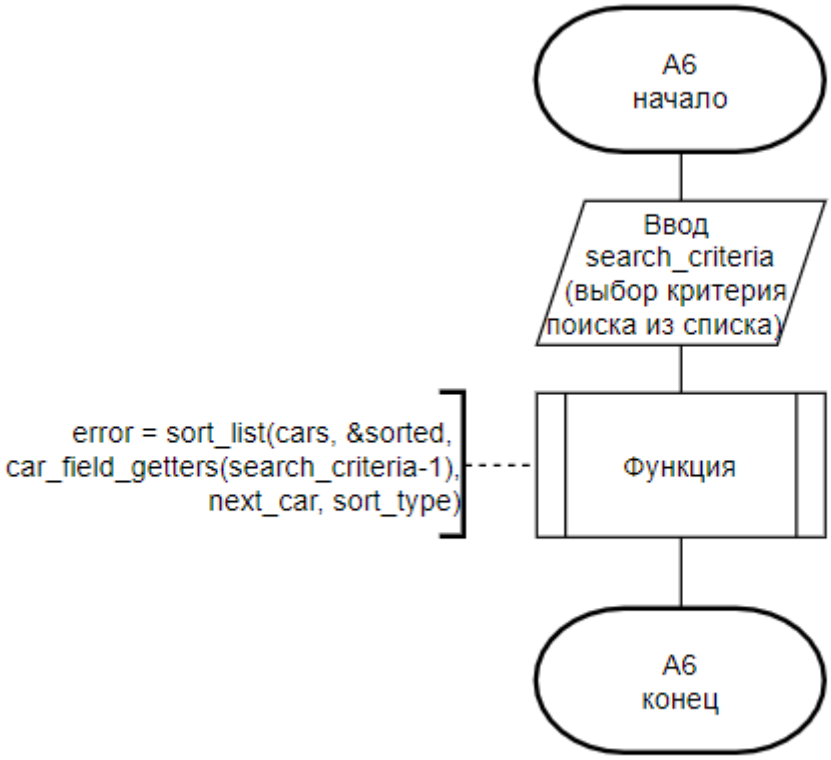

delete_list_element
начало
iter = iter.first
iter != 0
Да
iter == element
Да
prev = iter
iter =
следующий
элемент(iter)
iter == list.first
Да
list.first =
следующий
элемент(iter)
iter == list.last
Да
list.last = prev
prev != 0
Да
следующий
элемент(prev) =
следующий
элемент(iter)
list.count --
delete_list_element
конец

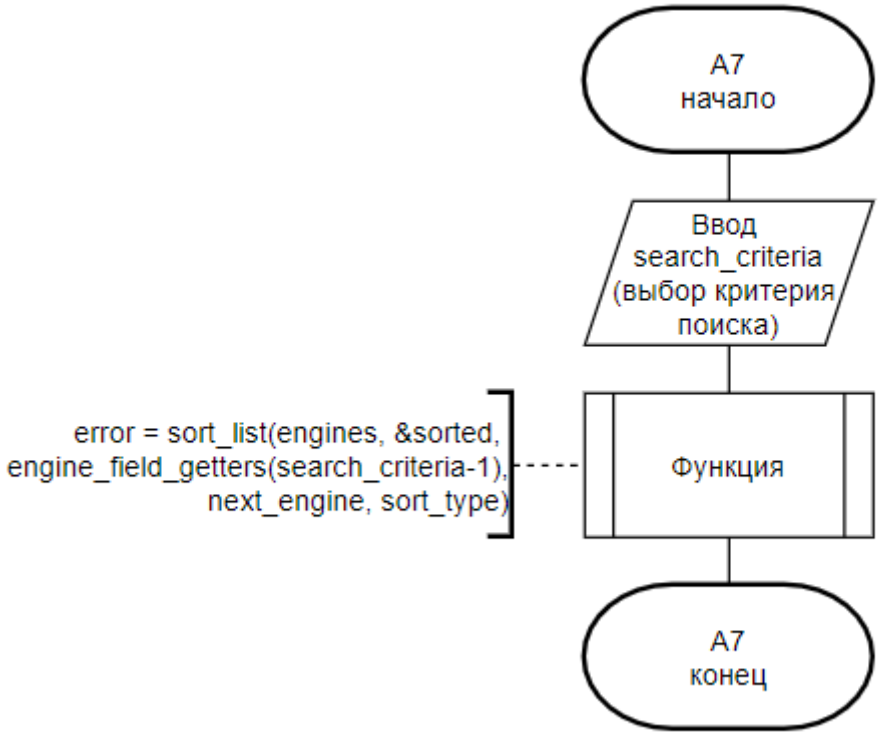

search_data
начало
Ввод
table_selection
(выбор таблицы
из списка)
table_selection
1
2
A5
Блок операций
A4
Блок операций
found == 0
Да
Вывод ошибки
Вывод error
search_data
конец

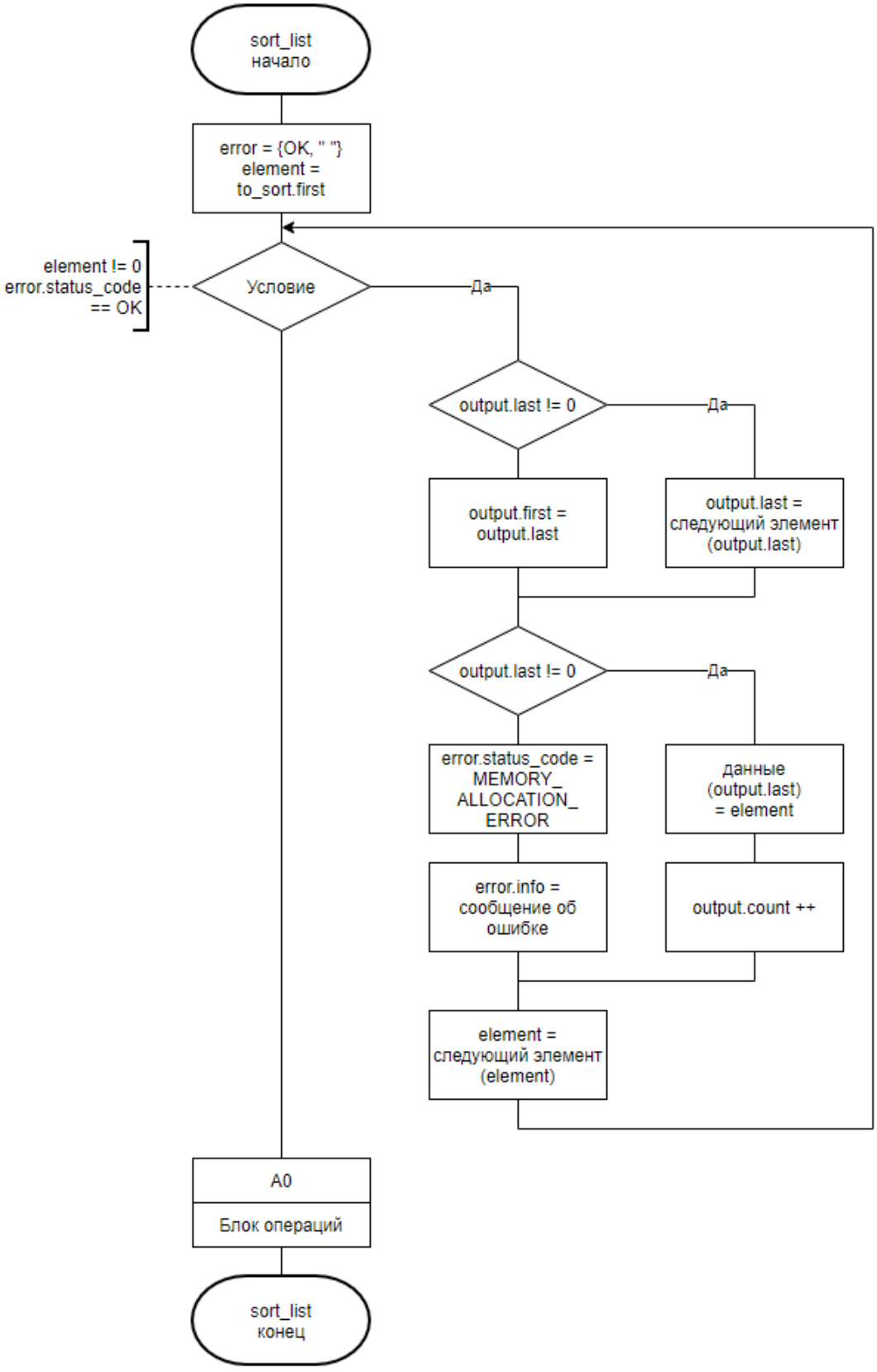

A4
начало
Ввод
search_criteria
(выбор критерия
поиска из списка)
Ввод buf
found = search(cars.
buf, value_size,
car_field_getters(search_criteria-1),
next_car, print_car, is_str)
Функция
A4
конец

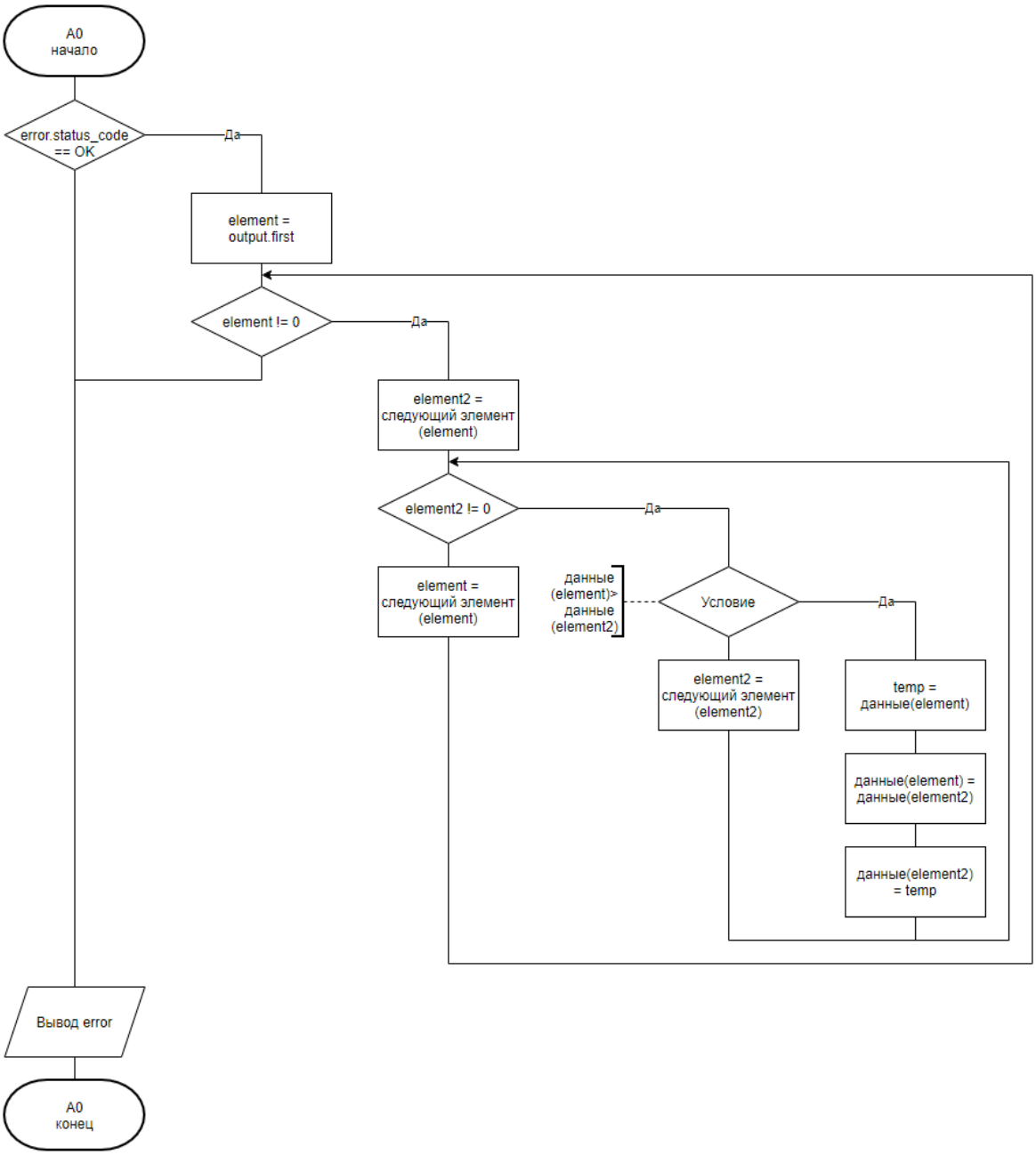

A5
начало
Ввод
search_criteria
(выбор критерия
поиска из списка)
Ввод buf
found = search(engines.
buf, value_size,
engine_field_getters(search_criteria-1),
next_engine, print_engine, is_str)
Функция
A5
конец

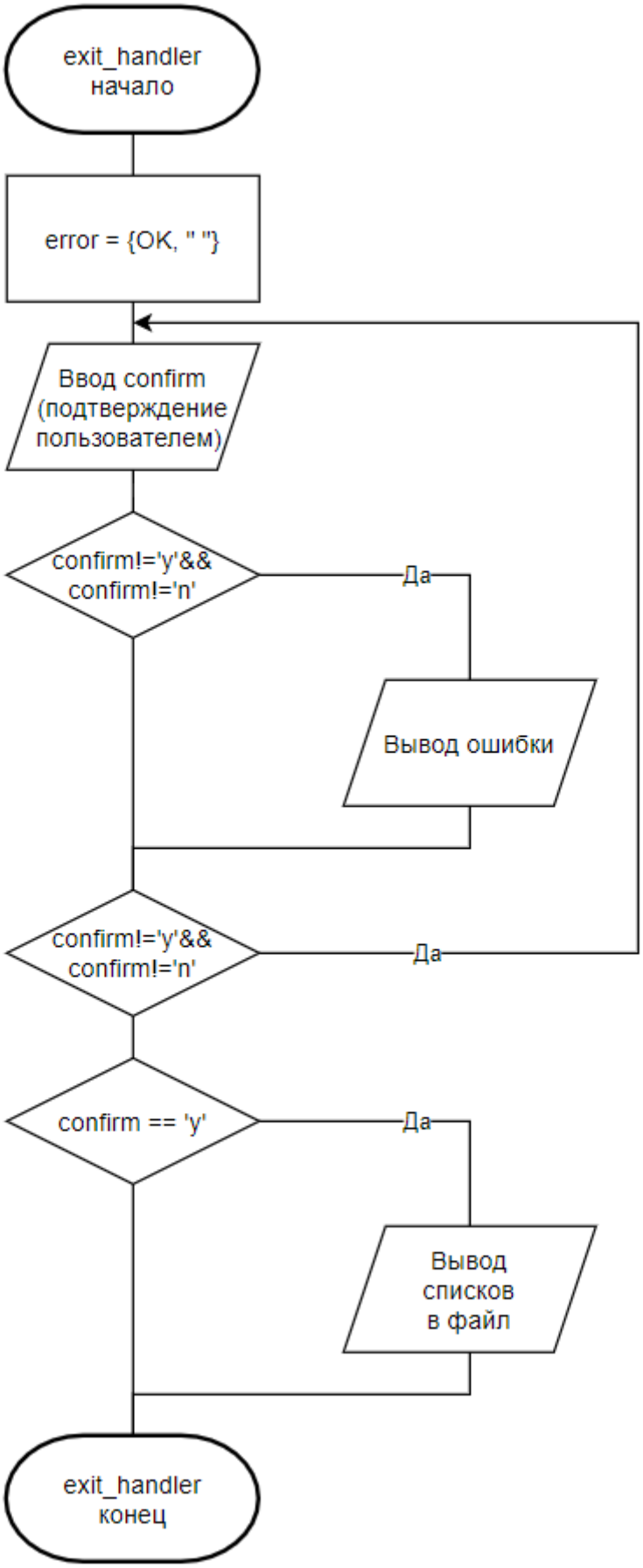

search
начало
iter = list.first
found = 0
iter != 0
Да
is_str != 0
Да
Совпадение
найдено
Да
Вывод
iter
found = 1
Совпадение
найдено
Да
Вывод
iter
found = 1
iter = следующий
элемент (iter)
Вывод found
search
конец


A6
начало
Ввод
search_criteria
(выбор критерия
поиска из списка)
error = sort_list(cars, &sorted,
car_field_getters(search_criteria-1),
next_car, sort_type)
Функция
A6
конец


A7
начало
Ввод
search_criteria
(выбор критерия
поиска)
error = sort_list(engines, &sorted,
engine_field_getters(search_criteria-1),
next_engine, sort_type)
Функция
A7
конец


sort_list
начало
error = {OK, " "}
element =
to_sort.first
element != 0
error.status_code
== OK
Условие
Да
output.last != 0
Да
output.first =
output.last
output.last =
следующий элемент
(output.last)
output.last != 0
Да
error.status_code =
MEMORY_
ALLOCATION_
ERROR
данные
(output.last)
= element
error.info =
сообщение об
ошибке
output.count ++
element =
следующий элемент
(element)
A0
Блок операций
sort_list
конец


A0
начало
error.status_code
== OK
Да
element =
output.first
element != 0
Да
element2 =
следующий элемент
(element)
element2 != 0
Да
element =
следующий элемент
(element)
данные
(element)>
данные
(element2)
Условие
Да
element2 =
следующий элемент
(element2)
temp =
данные(element)
данные(element) =
данные(element2)
данные(element2)
= temp
Вывод error
A0
конец


exit_handler
начало
error = {OK, " "}
Ввод confirm
(подтверждение
пользователем)
confirm!='y'&&
confirm!='n'
Да
Вывод ошибки
confirm!='y'&&
confirm!='n'
Да
confirm == 'y'
Да
Вывод
списков
в файл
exit_handler
конец

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.

```
Engines:
+-------+---------------------+-----------+-------+
|    ID |                Name |    Volume | Power |
+-------+---------------------+-----------+-------+
|     1 |               4K4C8 |      4.00 |   600 |
|     2 |                 DNW |      2.50 |   400 |
|     3 |            2JZ-GTE |      3.00 |   324 |
|     4 |          VR38DETT |      3.80 |   570 |
|     5 |               FA20 |      2.00 |   200 |
|     6 |              508PS |      5.00 |   575 |
|     7 |             S85B50 |      5.00 |   507 |
|     8 |               G6DU |      3.50 |   249 |
|     9 |            13B-REW |      1.30 |   280 |
|    10 |           RB26DETT |      2.60 |   280 |
|    11 |         M178DE40AL |      4.00 |   585 |
+-------+---------------------+-----------+-------+

Cars:
+-------+---------------------+-----------------------------------------+-----------+---------+---------------------+
|    ID |               Brand |                                   Model |      Cost |   Issue |              Engine |
+-------+---------------------+-----------------------------------------+-----------+---------+---------------------+
|     1 |                Audi |                                     RS6 |     18.50 |    2022 |               4K4C8 |
|     2 |                Audi |                                     RS3 |      9.60 |    2022 |                 DNW |
|     3 |              Toyota |                                   Supra |      5.50 |    1994 |            2JZ-GTE |
|     4 |              Nissan |                                     GTR |     12.90 |    2016 |          VR38DETT |
|     5 |              Toyota |                                    GT86 |      1.80 |    2014 |               FA20 |
|     6 |              Jaguar |                                  F-Type |      8.30 |    2017 |              508PS |
|     7 |                 BMW |                                      M6 |      2.30 |    2007 |             S85B50 |
|     8 |             Hyundai |                                  Staria |      5.20 |    2022 |               G6DU |
|     9 |               Mazda |                                    RX-7 |      2.50 |    1999 |            13B-REW |
|    10 |              Nissan |                                     GTR |      4.20 |    1999 |           RB26DETT |
|    11 |              Nissan |                                     GTR |      3.00 |    1993 |           RB26DETT |
|    12 |             Mercedes |                                     GTR |     11.90 |    2017 |         M178DE40AL |
+-------+---------------------+-----------------------------------------+-----------+---------+---------------------+
Press ENTER to continue...
```

```
Usage:
0. Show this help
1. Add new row to the end of database table
2. Edit one of existing rows in database
3. Remove one row from database
4. Print all database tables one by one
5. Search in tables by one of properties
6. Print one of tables sorted by selected property
7. Exit from program and save (or not) changes
Select operation: 5
Select table to search data in:
1. Cars
2. Engines
1
Select criteria to search by:
1. id
2. Brand
3. Model
4. Year of issue
5. Cost
6. Engine name
2
Enter value:Audi
|     1 |                Audi |                                     RS6 |     18.50 |    2022 |               4K4C8 |
|     2 |                Audi |                                     RS3 |      9.60 |    2022 |                 DNW |
Press ENTER to continue...
```

```
... database:
Usage:
0. Show this help
1. Add new row to the end of database table
2. Edit one of existing rows in database
3. Remove one row from database
4. Print all database tables one by one
5. Search in tables by one of properties
6. Print one of tables sorted by selected property
7. Exit from program and save (or not) changes
Select operation: 6
Select table to sort data:
1. Cars
2. Engines
1
Select criteria to sort by:
1. id
2. Brand
3. Model
4. Year of issue
5. Cost
6. Engine name
5
Sorted:
+-------+--------------------+----------------------------------------+-----------+----------+--------------------+
|    ID |              Brand |                                  Model |      Cost |    Issue |             Engine |
+-------+--------------------+----------------------------------------+-----------+----------+--------------------+
|     5 |             Toyota |                                   GT86 |      1.80 |     2014 |               FA20 |
|     7 |                BMW |                                     M6 |      2.30 |     2007 |            S85B50 |
|     9 |              Mazda |                                   RX-7 |      2.50 |     1999 |           13B-REW |
|    11 |             Nissan |                                    GTR |      3.00 |     1993 |           RB26DETT |
|    10 |             Nissan |                                    GTR |      4.20 |     1999 |           RB26DETT |
|     8 |            Hyundai |                                 Staria |      5.20 |     2022 |               G6DU |
|     3 |             Toyota |                                  Supra |      5.50 |     1994 |           2JZ-GTE |
|     6 |             Jaguar |                                 F-Type |      8.30 |     2017 |              508PS |
|     2 |               Audi |                                    RS3 |      9.60 |     2022 |               DNW |
|    12 |           Mercedes |                                    GTR |     11.90 |     2017 |         M178DE40AL |
|     4 |             Nissan |                                    GTR |     12.90 |     2016 |           VR38DETT |
|     1 |               Audi |                                    RS6 |     18.50 |     2022 |              4K4C8 |
+-------+--------------------+----------------------------------------+-----------+----------+--------------------+
Press ENTER to continue...
```

```
Usage:
0. Show this help
1. Add new row to the end of database table
2. Edit one of existing rows in database
3. Remove one row from database
4. Print all database tables one by one
5. Search in tables by one of properties
6. Print one of tables sorted by selected property
7. Exit from program and save (or not) changes
Select operation: 3
Select table to delete data from:
1. Cars
2. Engines
1
Enter id value (positive): 10
Deleted element:
|    10 |             Nissan |                                    GTR |      4.20 |     1999 |           RB26DETT |
Press ENTER to continue...
```

```
Usage:
0. Show this help
1. Add new row to the end of database table
2. Edit one of existing rows in database
3. Remove one row from database
4. Print all database tables one by one
5. Search in tables by one of properties
6. Print one of tables sorted by selected property
7. Exit from program and save (or not) changes
Select operation: 4

Engines:
+-------+--------------------+------------+-------+
|    ID |               Name |     Volume | Power |
+-------+--------------------+------------+-------+
|     1 |              4K4C8 |       4.00 |   600 |
|     2 |                DNW |       2.50 |   400 |
|     3 |            2JZ-GTE |       3.00 |   324 |
|     4 |           VR38DETT |       3.80 |   570 |
|     5 |               FA20 |       2.00 |   200 |
|     6 |              508PS |       5.00 |   575 |
|     7 |              S85B50 |       5.00 |   507 |
|     8 |               G6DU |       3.50 |   249 |
|     9 |            13B-REW |       1.30 |   280 |
|    10 |           RB26DETT |       2.60 |   280 |
|    11 |         M178DE40AL |       4.00 |   585 |
+-------+--------------------+------------+-------+

Cars:
+-------+--------------------+--------------------------------------------+------------+----------+--------------------+
|    ID |              Brand |                                      Model |       Cost |    Issue |             Engine |
+-------+--------------------+--------------------------------------------+------------+----------+--------------------+
|     1 |               Audi |                                        RS6 |      18.50 |     2022 |              4K4C8 |
|     2 |               Audi |                                        RS3 |       9.60 |     2022 |                DNW |
|     3 |             Toyota |                                      Supra |       5.50 |     1994 |            2JZ-GTE |
|     4 |             Nissan |                                        GTR |      12.90 |     2016 |           VR38DETT |
|     5 |             Toyota |                                       GT86 |       1.80 |     2014 |               FA20 |
|     6 |             Jaguar |                                     F-Type |       8.30 |     2017 |              508PS |
|     7 |                BMW |                                         M6 |       2.30 |     2007 |              S85B50 |
|     8 |            Hyundai |                                     Staria |       5.20 |     2022 |               G6DU |
|     9 |              Mazda |                                       RX-7 |       2.50 |     1999 |            13B-REW |
|    11 |             Nissan |                                        GTR |       3.00 |     1993 |           RB26DETT |
|    12 |           Mercedes |                                        GTR |      11.90 |     2017 |         M178DE40AL |
+-------+--------------------+--------------------------------------------+------------+----------+--------------------+
Press ENTER to continue...
```

Заключение.

Используемые заголовочные файлы стандартной библиотеки:

1. stdio.h
   1.1. fputs – записывает строку в файл
   1.2. fputc – записывает символ в файл
   1.3. scanf – считывает данные
   1.4. fgetc – считывает символ из файла
   1.5. fopen – открывает файл
   1.6. fseek – ищет в файле
   1.7. printf – выводит аргументы
   1.8. puts – выводит строку
   1.9. fwrite – записывает объекты в файл
   1.10. fgets – считывает символы из файла
   1.11. sscanf – считывает данные из массива
2. stdlib.h
   2.1. system – передает команду командному процессору
   2.2. free – освобождает память
   2.3. malloc – возвращает адрес на первый байт области памяти
   2.4. realloc – изменяет величину выделенной памяти на новую
3. string.h
   3.1. strstr – ищет по строке
   3.2. memcmp – сравнивает символы
   3.3. strcmp – сравнивает строки
   3.4. strcpy – копирует из одной строки в другую
   3.5. strlen – считает длину строки
4. stdarg.h
   4.1. va_arg – осуществляет передачу функции переменного числа параметров
   4.2. va_list – определяет тип va_list
   4.3. va_start – осуществляет передачу функции переменного числа параметров
5. stddef.h
   5.1. size_t – определяет тип size_t
6. float.h
   6.1. DBL_MAX_10_EXP – максимальное целое число, при котором число 10, возведенное в степень этого числа, является представимым числом с плавающей запятой.
   6.2. 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
   7.1. ULONG_MAX – максимальное значение для беззнакового long int
8. math.h
   8.1. log10 – считает десятичный логарифм числа

8.2. ceil – считает наименьшее целое, которое не меньше заданного числа

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