Министерство науки и образования РФ
Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)»

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

Пояснительная записка
к курсовой работе
по дисциплине «Программирование»
на тему: «Разработка электронной картотеки»

Выполнил студент гр. 9308 Яловега Н.В

Проверил к.т.н., доцент Перязева Ю.В

Санкт-Петербург, 2020

Оглавление

Введение

Целью курсовой работы является полное решение содержательной задачи (содержательная и формальная постановка задачи, спецификация, включая описание диалога, выбор метода решения и структур данных, разработка алгоритма, программная реализация, тестирование и отладка, документирование). Задача: создание электронной картотеки музыкальных произведений

1. Задание

Создать электронную картотеку, хранящуюся на диске, и программу, обеспечивающую взаимодействие с ней.

Программа должна выполнять следующие действия:

- занесение данных в электронную картотеку;
- внесение изменений (исключение, корректировка, добавление);
- поиск данных по различным признакам;
- сортировку по различным признакам;
- вывод результатов на экран и сохранение на диске.

Выбор подлежащих выполнению команд должен быть реализован с помощью основного меню и вложенных меню.

Задача должна быть структурирована и отдельные части должны быть оформлены как функции.

Исходные данные должны вводиться с клавиатуры. В процессе обработки картотека должна храниться в памяти компьютера в виде списков и массивов структур, связанных указателями. Типы списков и структур выбираются исходя из предметной области.

Картотека составляется по выбранной предметной области.

В программе должно быть реализовано простейшее меню. Выполнение программы должно быть многократным по желанию пользователя. Данные первоначально считываются из файла (файлов), в процессе работы данные вводятся с клавиатуры.

2. Постановка задачи и описание решения

Выбранная предметная область - музыкальная картотека.

Исходя из выбранной предметной области, были выбраны следующие структуры:

Таблица 1. Описание полей структуры list_info

Имя поля	**Тип**	**Назначение**
artist	char*	Имя исполнителя
title	char*	Название композиции
album	char*	Альбом
genre	int	Жанр
year	int	Год выхода композиции
number	int	Номер композиции в альбоме

Поля вспомогательной структуры “узла”:

Таблица 2. Описание полей структуры list_node

Имя поля	**Тип**	**Назначение**
data	struct list_info	Информация об узле
next	struct list_node*	Указатель на адрес следующего узла
previous	struct list_node*	Указатель на адрес предыдущего узла

Поля вспомогательной структуры “списка”:

Таблица 3. Описание полей структуры list

Имя поля	**Тип**	**Назначение**
size	int	Размер списка
head	struct list_node*	Указатель на первый элемент в списке
tail	struct list_node*	Указатель на последний элемент в списке

Поля artist, title, album должны содержать строки не длинее 21 символа(включая нуль терминатор). Поле genre должно определять жанр программы.

Доступные жанры:

1 - Классическая музыка

2 – Поп

3 – Танцевальная

4 – Электронная

5 – Фолк

6 – Рок

7 – Хип-хоп

Поле year должно содержать число от 1 до 3000, а поле number — число от 1 до 100.

Для решения поставленной задачи необходимы были файлы с работой линейными списками, а также с работой файлов. Исходный код программы был разбит на модули как показано в таблице 4.

Таблица 4. Описание файлов

Файл	Описание
main.c	Содержит функцию main
menu.h	Содержит прототипы функций и предварительные объявления для menu.c
menu.c	Содержит функции вывода меню, интерфейса
constants.h	Содержит прототипы функций и предварительные объявления для constants.c
constants.c	Сожержит константы проекта
help_func.h	Содержит прототипы функций и предварительные объявления для help_func.c
help_func.c	Вспомогательные функции
dlist.h	Содержит прототипы функций и предварительные объявления для dlist.c
dlist.c	Функции для работы с двусвязным линейным списком

Для решения задачи было сформировано меню с пунктам

1 — Подсказка

2 — Ввести

- 2.1 Ввести с консоли
  - 2.1.1 Добавить в начало
  - 2.1.2 Добавить в конец
  - 2.1.3 Вставить в n — позицию
- 2.2 Считать из файла

3 — Действия со списком

- 3.1 Изменение карточки
- 3.2 Сортировка списка
- 3.3 Реверс списка
- 3.4 Случайное перемешивание
- 3.5 Перестановка двух элементов
- 3.6 Удаление списка

3 — Поиск

4 — Вывод

- 4.1 Вывод полного списка
- 4.2 Вывод одного элемента
- 4.3 Сохранение в файл

0 – Выход

Пункты меню 3, 4, 5 не должны показываться, пока пользователь не введет все данные (пункт 2). Если пользователь выберет несуществующий или скрытый пункт меню, то нужно ему сообщить об этом и предложить выбрать пункт меню заново.

3. Функции

3.1. Функции файла main.c

3.1.1. Главная функция

Назначение:

Является точкой входа в программу.

Прототип:

int main()

Пример вызова:

main();

Описание переменных:

Описание переменных приведено в таблице 5.

Таблица 5. Описание переменных главной функции

Имя переменной	Тип	Назначение
music_list	LIST*	Исходный двусвязный список
variant	int	Выбранный пользователем пункт меню
i	int	Счетчик цикла
flags	unsigned char	Флаги доступа к пунктам меню

3.2. Функции файла help_func.c

3.2.1. Функция get_string

Назначение:

Функция чтения строки

Прототип:

char *get_string(unsigneg char);

Пример вызова:

music.artist = get_string(1);

Описание переменных:

Описание переменных приведено в таблице 6.

Таблица 6. Описание переменных функции get_string

Вид переменной	Имя переменной	Тип	Назначение
Формальная	type	unsigned char	Переменная для вывода подсказки при вводе строки.
Локальная	s	char*	Указатель на первый символ динамической строки
Локальная	k	int	Текущий символ
Локальная	i	int	Счетчик цикла

3.2.2. Функция get_genre

Назначение:

Получение номера жанра из списка всех возможных.

Прототип:

int get_genre();

Пример вызова:

music.genre = get_genre();

Описание переменных:

Описание переменных приведено в таблице 7.

Таблица 7. Описание переменных функции get_genre

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Локальная	g	int	Число, вводимое пользователем

3.2.3. Функция get_year

Назначение:

Получение года выхода музыкальной композиции.

Прототип:

int get_year();

Пример вызова:

music.year = get_year();

Описание переменных:

Описание переменных приведено в таблице 8.

Таблица 8. Описание переменных функции get_year

Вид переменной	Имя переменной	Тип	Назначение
Локальная	y	int	Число, вводимое пользователем

3.2.4. Функция get_number

Назначение:

Получение номера композиции в музыкальном альбоме.

Прототип:

int get_number();

Пример вызова:

music.number = get_number();

Описание переменных:

Описание переменных приведено в таблице 9.

Таблица 9. Описание переменных функции get_number

Вид переменной	Имя переменной	Тип	Назначение
Локальная	n	int	Число, вводимое пользователем

3.2.5 Функция print_array

Назначение:

Вывод константного массива констант строк

Прототип:

void print_array(const char * const *, unsigneg);

Пример вызова:

print_array(sort_array, SORT_NUM);

Описание переменных:

Описание переменных приведено в таблице 10.

Таблица 10. Описание переменных функции print_array

Вид переменной	Имя переменной	Тип	Назначение
Локальная	i	int	Счетчик цикла

3.2.6 Функция clean_stdin

Назначение:

Замена функции очистки входного потока (fflush)

Прототип:

void clean_stdin();

Пример вызова:

clean_stdin();

Описание переменных:

Описание переменных приведено в таблице 11.

Таблица 11. Описание переменных функции clean_stdin

Вид переменной	Имя переменной	Тип	Назначение
Локальная	c	char	Лишний символ во входном потоке

3.2.7 Функция print_msg

Назначение:

Вывод важной для пользователя информации на экран. (Например, информация об ошибке, приветствие)

Прототип:

void print_msg(char*);

Пример вызова:

print_msg(“Some text”);

3.2.8 Функция print_header

Назначение:

Вывод шапки таблицы

Прототип:

void print_header();

Пример вызова:

print_header();

3.2.9 Функция pause

Назначение:

Сохранение информации на экране, пока пользователь не захочет продолжить выполнение программы.

Прототип:

void pause();

Пример вызова:

pause();

3.3. Функции файла dlist.c

3.3.1. Функция create_list

Назначение:

Создание двусвязного линейного списка.

Прототип:

LIST *create_list();

Пример вызова:

music_list = create_list();

Описание переменных:

Описание переменных приведено в таблице 12.

Таблица 12. Описание переменных функции create_list

Вид переменной	Имя переменной	Тип	Назначение
Локальная	temp	LIST*	Новый список

3.3.2. Функция push

Назначение:

Добаление нового элемента в начало списка. Сложность O(1)

Прототип:

void push(LIST *, TRACK)

Пример вызова:

push(music_list, track);

Описание переменных:

Описание переменных приведено в таблице 13.

Таблица 13. Описание переменных функции push

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Формальная	data	TRACK	Стуктура, которую нужно добавить в список
Локальная	new_node	NODE*	Новый элемент списка

3.3.3. Функция pop

Назначение:

Удаление первого элемента из списка. Сложность O(1)

Прототип:

void pop(LIST*);

Пример вызова:

pop(list);

Описание переменных:

Описание переменных приведено в таблице 14.

Таблица 14. Описание переменных функции pop

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Локальная	previous	NODE*	Второй элемент списка

3.3.4. Функция append

Назначение:

Добавление нового элемента в конец списка. Сложность O(n)

Прототип:

void append(LIST *, TRACK);

Пример вызова:

append(list, music);

Описание переменных:

Описание переменных приведено в таблице 15.

Таблица 15. Описание переменных функции append

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	LIST*	Указатель на голову списка
Формальная	data	TRACK	Стуктура, которую нужно добавить в список
Локальная	new_node	NODE*	Новый элемент списка

3.3.5. Функция get

Назначение:

Получение элемента списка. Сложность O(n)

Прототип:

NODE *get(LIST *, unsigned)

Пример вызова:

node = get(list, 2);

Описание переменных:

Описание переменных приведено в таблице 16.

Таблица 16. Описание переменных функции get

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	LIST*	Голова списка
Формальная	index	unsigned	Номер элемента списка
Локальная	i	unsigned	Счетчик цикла

3.3.6. Функция swap

Назначение:

Поменять местами два элемента списка.

Прототип:

void swap (LIST *, unsigned, unsigned)

Пример вызова:

swap(list, i, i+1);

Описание переменных:

Описание переменных приведено в таблице 17.

Таблица 17. Описание переменных функции swap

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	LIST*	Список
Формальная	pos1	unsigned	Позиция одного элемента
Формальная	pos2	unsigned	Позиция другого элемента
Локальная	tmp	TRACK	Временная структура для сохранения информации
Локальная	first	NODE*	Указатель на первый элемент
Локальная	last	NODE*	Указатель на второй элемент

3.3.7. Функция sort

Назначение:

Сортировка элементов списка методом пузырька

Прототип:

void sort(LIST *, unsigned)

Пример вызова:

sort(music_list, 1);

Описание переменных:

Описание переменных приведено в таблице 18.

Таблица 18. Описание переменных функции sort

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Формальная	type	unsigned	Тип сортируемого значения
Локальная	n	int	Длина списка
Локальная	i,j	int	Счетчики

3.3.8. Функция length

Назначение:

Получение длины списка. Сложность O(1)

Прототип:

int length(LIST *)

Пример вызова:

n = length(list);

Описание переменных:

Описание переменных приведено в таблице 19.

Таблица 19. Описание переменных функции length

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	LIST*	Список

3.3.9. Функция print_list

Назначение:

Вывод информации о музыкальной композиции в виде таблицы.

Прототип:

void print_list(LIST *);

Пример вызова:

print_list(list);

Описание переменных:

Описание переменных приведено в таблице 20.

Таблица 20. Описание переменных функции print_list

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Локальная	count	unsigned	Счетчик цикла

3.3.10. Функция memory_clear

Назначение:

Очистка памяти полей структур.

Прототип:

void memory_clear(TRACK*);

Пример вызова:

memory_clear(head->data);

Описание переменных:

Описание переменных приведено в таблице 21.

Таблица 21. Описание переменных функции memory_clear

Вид переменной	Имя переменной	Тип	Назначение
Формальная	t	TRACK*	Указатель на структуру, у которой нужно очистить память у полей.

3.3.11. Функция get_database

Назначение:

Добавление в список структур, информация для которых содержится в файле. Считывает строку из файла, которую потом передает в функцию для разбивания строки на части. После чего данные части добавляются в структуру.

Прототип:

void get_database(NODE **, char*, char);

Пример вызова:

get_database(&head, path, ‘;’);

Описание переменных:

Описание переменных приведено в таблице 22.

Таблица 22. Описание переменных функции get_music_data

Вид переменной	Имя переменной	Тип	Назначение
Формальная	head	NODE**	Указатель на голову списка
Формальная	filename	char*	Указатель на первый символ строки, содержащая имя файла
Формальная	separator	char	Символ-разделитель в файле с информацией
Локальная	df	FILE*	Дескриптор файла
Локальная	s1	char*	Строка файла
Локальная	s2	char**	Массив разделенных строк
Локальная	slen	int	Реальный размер считанной из файла строки
Локальная	flag	int	Флаг, для очистки памяти

3.3.12. Функция simple_split

Назначение:

Функция разбиения строки на массив строк

Прототип:

char **simple_split(char *, int, char)

Пример вызова:

s2 = simple_split(s1,slen,separator);

Описание переменных:

Описание переменных приведено в таблице 23.

Таблица 23.Описание переменных функции simple_split

Вид переменной	Имя переменной	Тип	Назначение
Формальная	str	char*	Строка для разделения
Формальная	length	int	Длина строки
Формальная	sep	char	Символ-разделитель в файле с информацией
Локальная	str_array	char**	Массив разделенных строк
Локальная	i,j,m,k	int	Счетчики

3.3.13. Функция fill_node

Назначение:

Функция заполнения структуры данными, полученными при разбиении строки файла на части

Прототип:

TRACK fill_node(char **)

Пример вызова:

p = fill_node(s2);

Описание переменных:

Описание переменных приведено в таблице 24.

Таблица 24.Описание переменных функции simple_split

Вид переменной	Имя переменной	Тип	Назначение
Формальная	s2	char**	Массив разделенных строк
Локальная	p	TRACK	Структура,в которую добавляется информация
Локальная	len1, len2, len3	int	Длины строк

3.3.14. Функция delete_list

Назначение:

Удаление списка.

Прототип:

void delete_list(LIST **);

Пример вызова:

delete_list(&music_list);

Описание переменных:

Описание переменных приведено в таблице 25.

Таблица 25. Описание переменных функции delete_list

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	LIST**	Указатель на список
Локальная	previous	NODE*	Указатель на элемент списка
Локальная	temp_node	NODE*	Указатель на элемент списка

3.4. Функции файла menu.c

3.4.1. Функция print_menu

Назначение:

Вывод меню на экран.

Прототип:

void print_menu(unsigned char, unsigned char*);

Пример вызова:

print_menu(1, 0);

Описание переменных:

Описание переменных приведено в таблице 26.

Таблица 26. Описание переменных функции print_menu

Вид переменной	Имя переменной	Тип	Назначение
Формальная	menu	unsigned char	Номер меню
Формальная	flags	unsigned char*	Одномерный массив флагов, для вывода только доступных пунктов меню

3.4.2. Функция input_menu

Назначение:

Меню выбора для ввода данных

Прототип:

void input_menu(LIST *list);

Пример вызова:

input_menu(music_list);

Описание переменных:

Описание переменных приведено в таблице 27.

Таблица 27. Описание переменных функции input_menu

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	LIST*	Список
Локальная	variant	int	Переменная для выбора пункта меню
Локальная	path	char*	Путь до файла с данными
Локальная	answer	char	Ответ пользователя на вопрос о вводе еще одной структуры

3.4.3. Функция output_menu

Назначение

Меню выбора для вывода данных

Прототип:

void output_menu(LIST *);

Пример вызова:

output_menu(LIST* music_list);

Описание переменных:

Описание переменных приведено в таблице 28.

Таблица 28. Описание переменных функции output_menu

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Локальная	variant	int	Переменная для выбора пункта меню
Локальная	n	int	Номер элемента в списке, который нужнов вывести

3.4.4. Функция search_menu

Назначение

Меню выбора для поиска данных

Прототип:

void search_menu(LIST *);

Пример вызова:

search_menu(LIST* music_list);

Описание переменных:

Описание переменных приведено в таблице 29.

Таблица 29. Описание переменных функции search_menu

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Локальная	variant	int	Переменная для выбора пункта меню
Локальная	type	unsigned	Тип сортировки при выводе
Локальная	n	int	Номер элемента в списке, который нужнов вывести

3.4.5. Функция edit_menu

Назначение

Меню выбора для редактирования данных

Прототип:

void edit_menu(LIST **);

Пример вызова:

edit_menu(LIST** music_list);

Описание переменных:

Описание переменных приведено в таблице 30.

Таблица 30. Описание переменных функции edit_menu

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST**	Указатель на список
Локальная	variant	int	Переменная для выбора пункта меню
Локальная	first	int	Номер элемента для перестановки
Локальная	second	int	Номер элемента для перестановки

3.4.6. Функция edit_info_menu

Назначение

Меню редактирования информации.

Прототип:

void edit_info_menu(LIST *);

Пример вызова:

edit_info_menu(LIST* music_list);

Описание переменных:

Описание переменных приведено в таблице 31.

Таблица 31. Описание переменных функции edit_info_menu

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Локальная	change_int	int	Переменная для хранения нового значения целого типа
Локальная	change_str	char*	Переменная для хранения нового значения строки
Локальная	temp_node	NODE*	Временный узел
Локальная	variant1,variant2	int	Переменные для выбора пункта меню

3.4.6. Функция sort_menu

Назначение

Меню сортировки информации.

Прототип:

void sort_menu(LIST *);

Пример вызова:

sort_menu(LIST* music_list);

Описание переменных:

Описание переменных приведено в таблице 32.

Таблица 32. Описание переменных функции sort_menu

Вид переменной	**Имя переменной**	**Тип**	**Назначение**
Формальная	list	LIST*	Список
Локальная	type	int	Поле сортировки
Локальная	rev	int	Тип сортировки

3.4.7. Функция help

Назначение:

Получение подсказки

Прототип:

void help();

Пример вызова:

help();

3.4.8. Функция get_music_data

Назначение:

Получение струтуры с информацией о композиции из консоли

Прототип:

TRACK get_music_data();

Пример вызова:

p = get_music_data();

Описание переменных:

Описание переменных приведено в таблице 33.

Таблица 33. Описание переменных функции get_music_data

Вид переменной	Имя переменной	Тип	Назначение
Локальная	music	TRACK	Структура с информацией о композиции

4. Контрольные примеры

Исходные данные (файл data.csv)

ESH;Nothing at All;Nothing at All;1;2019;2

Modern Talking;Brother Louie;Ready For Romance;1;1986;2

DOP;Melancholia;Email From a Beetle;3;2017;3

Sean Tyas;Chrome;Solo Vol.I;2;2019;3

Waterflame;Clutterfunk;Clutterfunk;1;2012;3

Waterflame;Jumper;Jumper;1;2005;3

Zamay;Air Tuapse;Lust Hero 2;1;2020;6

Waterflame;Time Machine;Time Machine;1;2009;3

Ociya;Gravity Knots;Powers Of Ten;6;2020;3

The White Stripes;Seven Nation Army;Elephant;1;2002;5

Imagine Dragons;Believer;Evolve;4;2017;5

Egor Letov;Against;Battle Incentive;15;1988;5

Slava KPSS;Man Whithout Skin;Buter Brodsky;3;2019;6

Egor Letov;We ice;Totalitarianism;1;1987;5

ACDC;Highway to Hell;Highway to Hell;1;1976;5

Mudvayne;Happy?;Lost and Found;3;2005;5

Eminem;Not Afraid;Recovery;7;2010;6

ForeverBound;Stereo Madness;Stereo Madness;1;2012;3

1) Отсортировать по году по возрастанию:

ACDC;Highway to Hell;Highway to Hell;1;1976;5

Modern Talking;Brother Louie;Ready For Romance;1;1986;2

Egor Letov;We ice;Totalitarianism;1;1987;5

Egor Letov;Against;Battle Incentive;15;1988;5

The White Stripes;Seven Nation Army;Elephant;1;2002;5

Waterflame;Jumper;Jumper;1;2005;3

Mudvayne;Happy?;Lost and Found;3;2005;5

Waterflame;Time Machine;Time Machine;1;2009;3

Eminem;Not Afraid;Recovery;7;2010;6

Waterflame;Clutterfunk;Clutterfunk;1;2012;3

ForeverBound;Stereo Madness;Stereo Madness;1;2012;3

DOP;Melancholia;Email From a Beetle;3;2017;3

Imagine Dragons;Believer;Evolve;4;2017;5

ESH;Nothing at All;Nothing at All;1;2019;2

Sean Tyas;Chrome;Solo Vol.I;2;2019;3

Slava KPSS;Man Whithout Skin;Buter Brodsky;3;2019;6

Zamay;Air Tuapse;Lust Hero 2;1;2020;6

Ociya;Gravity Knots;Powers Of Ten;6;2020;3

2) Найти все элементы с жанром 3 и исполнителем Waterflame

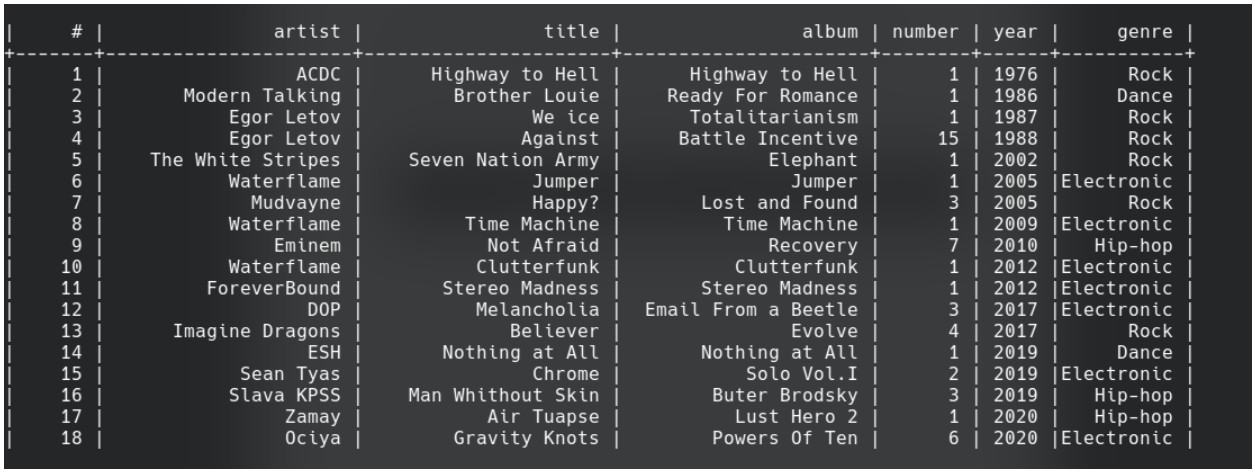
Waterflame;Time Machine;Time Machine;1;2009;3

Waterflame;Clutterfunk;Clutterfunk;1;2012;3

Waterflame;Jumper;Jumper;1;2005;3

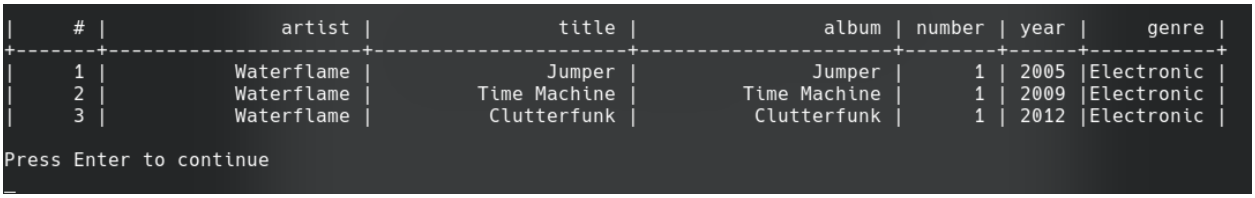
5. Тестирование работы программы

Для тестирования программы были использованны тесты, которые описаны в пункте Контрольные примеры. Результат совпадает с ожидаемым.



#	artist	title	album	number	year	genre
1	ACDC	Highway to Hell	Highway to Hell	1	1976	Rock
2	Modern Talking	Brother Louie	Ready For Romance	1	1986	Dance
3	Egor Letov	We ice	Totalitarianism	1	1987	Rock
4	Egor Letov	Against	Battle Incentive	15	1988	Rock
5	The White Stripes	Seven Nation Army	Elephant	1	2002	Rock
6	Waterflame	Jumper	Jumper	1	2005	Electronic
7	Mudvayne	Happy?	Lost and Found	3	2005	Rock
8	Waterflame	Time Machine	Time Machine	1	2009	Electronic
9	Eminem	Not Afraid	Recovery	7	2010	Hip-hop
10	Waterflame	Clutterfunk	Clutterfunk	1	2012	Electronic
11	ForeverBound	Stereo Madness	Stereo Madness	1	2012	Electronic
12	DOP	Melancholia	Email From a Beetle	3	2017	Electronic
13	Imagine Dragons	Believer	Evolve	4	2017	Rock
14	ESH	Nothing at All	Nothing at All	1	2019	Dance
15	Sean Tyas	Chrome	Solo Vol.I	2	2019	Electronic
16	Slava KPSS	Man Whithout Skin	Buter Brodsky	3	2019	Hip-hop
17	Zamay	Air Tuapse	Lust Hero 2	1	2020	Hip-hop
18	Ociya	Gravity Knots	Powers Of Ten	6	2020	Electronic

Рисунок 1: Пример 1



#	artist	title	album	number	year	genre
1	Waterflame	Jumper	Jumper	1	2005	Electronic
2	Waterflame	Time Machine	Time Machine	1	2009	Electronic
3	Waterflame	Clutterfunk	Clutterfunk	1	2012	Electronic

Press Enter to continue

Рисунок 2: Пример 2

6. Исходный код программы

6.1. main.c

```
#include "menu.h"

int main()
{
    LIST *musiclist = NULL;

    int variant,
        i;

    enum
    /*
        Пункты главного меню
    */
    {
        EXIT,
        HELP,
        INPUT_DATA,
        EDIT_DATA,
        SEARCH_DATA,
        PRINT_DATA,
        MENU_NUM
    };
```

```
    unsigned char flags[MENU_NUM];

    for (i = 0; i < MENU_NUM; i++)
        flags[i] = 1;

    print_msg(HELLO);
    do
    {
        if (is_empty(musiclist))
            flags[EDIT_DATA] = flags[SEARCH_DATA] =
flags[PRINT_DATA] = 0;

        print_menu(MAIN_MENU, flags);
        save_scanf(&variant);

        switch (variant)
        {
            case HELP:
                help();
            break;
            case INPUT_DATA:
                if (musiclist == NULL)
                    musiclist = create_list();

                input_menu(musiclist);

                if (musiclist->head != NULL)
```

```
                    flags[EDIT_DATA] = flags[SEARCH_DATA] =
flags[PRINT_DATA] = 1;
            break;
            case EDIT_DATA:
                if (flags[EDIT_DATA])
                    edit_menu(&musiclist);
                else
                    print_msg(MENU_ERROR);
            break;
            case SEARCH_DATA:
                if (flags[SEARCH_DATA])
                    search_menu(musiclist);
                else
                    print_msg(MENU_ERROR);
            break;
            case PRINT_DATA:
                if (flags[PRINT_DATA])
                    output_menu(musiclist);
                else
                    print_msg(MENU_ERROR);
            break;
            case EXIT:
                flags[EXIT] = 0;
            break;
            default:
                print_msg(MENU_ERROR);
```

```
            break;
        }

    }
    while(flags[EXIT]);
    delete_list(&musiclist);
    return 0;
}
```

6.2. constants.h

```
/*
    Контанты проекта
*/

#ifndef CONSTANTS_H
#define CONSTANTS_H

// константа очистки консоли
#ifdef __linux__
    #define CLEAR "clear"
#else
    #define CLEAR "cls"
#endif

// ------------------------- Константы-Ошибки
------------------------

#define HELLO "Hello, user.\nThis program will help you
create your music library."
#define MENU_ERROR "Sorry, but there is no such menu item.\
nTry to select another item."
#define FILE_ERROR "Sorry, error of file reading. Try
another file."
#define FILE_SUCCESS "Successful file writing!"
#define LIST_DELETE "List of music was successfully delete"
#define INDEX_ERROR "Number can't be less than 0 or greater
than length"
```

```
#define SEARCH_ERROR "Sorry, but there are no elements that satisfy these conditions"

#define INPUT_ERROR "\nSorry, but you should enter number, not string"


// ------------------------- Прочие контанты -------------------------

#define GENRES_NUM 7 // Количество жанров музыки

#define SORT_NUM 6 // Количество видов сортировки музыки

#define MAXLEN 21 // Максимальная длина строки (+ нуль-терминатор)

#define MAXSTR 150 // Максимальная строка файла


extern const char * const genres_array[GENRES_NUM];


#endif
```

6.3. constants.c

```
#include "constants.h"

const char * const genres_array[GENRES_NUM] =
/*
    Доступные в программе жанры музыки
*/
{
    "Classical",
    "Pop",
    "Dance",
    "Electronic",
    "Folk",
    "Rock",
    "Hip-hop"
};
```

6.4. help_func.h

```
#ifndef HELP_FUNC_H
#define HELP_FUNC_H

void print_array(const char * const *, unsigned);
void clean_stdin();
void pause();
void print_msg(char *);
void print_header();
void clear_str_array(char **, int);
char *get_string(unsigned char);
int get_number();
int get_genre();
int get_year();
char save_scanf(int*);

#endif
```

6.5. help_func.c

```
#include <stdio.h>
#include <stdlib.h>
#include "constants.h"

void clean_stdin()
/**
 * @brief Замена функции очистки входного потока (fflush)
 */
{
    char c;
    while ( scanf("%c", &c) == 1 && c != '\n');
}

void print_array(const char * const array[], unsigned len)
/**
 * @brief Вывод массива
 */
{
    int i;

    for (i = 0; i < len; i++)
        printf("%d. %s \n", i, array[i]);
}
```

```
void clear_str_array(char **str, int n)
/**
 * @brief Очистка памяти
 */
{
    int i;
    for(i = 0;i < n;i++)
    {
        free(str[i]);
        str[i]=NULL;
    }
}

void pause()
/**
 * @brief Замена системного ожидания
 */
{
    puts("\nPress Enter to continue");
    getchar();
}

void print_msg(char *information)
/**
 * @brief Вывод на экран информации об ошибке/о
вспомогательной информации
 * ---
```

```
 * @param char *information - текст информации/ошибки
 */
{
    system(CLEAR);
    puts(information);
    pause();
}

void print_header()
/**
 *  @brief Вывод шапки таблицы
 */
{
    puts("");
    printf("|%6s |%21s |%21s |%21s |%7s |%5s |%10s |\n",
"#", "artist", "title", "album", "number", "year",
"genre");
    printf("+-------+----------------------
+----------------------+----------------------+--------
+------+-----------+\n");
}

char *get_string(unsigned char type)
/**
 *  @brief Получение строки из терминала
 */
{
    int i, k;
```

```
    char *s = NULL;


    if (type == 1)
        printf("Enter artist of track (max string length:
%d): ", MAXLEN);
    else if (type == 2)
        printf("Enter title of track (max string length:
%d): ", MAXLEN);
    else if (type == 3)
        printf("Enter album (max string length: %d): ",
MAXLEN);


    i = 0;
    s = malloc(sizeof(char));
    while(((k = getchar()) != '\n') && (i < MAXLEN))
    {
        s[i] = k;
        s = realloc(s, (1+(++i))*sizeof(char));
    }
    s[i] = '\0';


    if (i == MAXLEN)
        clean_stdin();
    return s;
}



int get_year()
```

```
/**
 *  @brief Получение года из терминала
 */
{
    int y;
    int good;

    good = 1;
    do
    {
        printf("Enter the year of track [1,3000]: ");
        good = scanf("%d", &y);
        clean_stdin();
        if (good != 1)
            puts(INPUT_ERROR);
        else if (y < 0)
            puts("\nSorry, but year should be great than
0");
        else if (y > 3000)
            puts("\nSorry, but year should be less than
3000");
    }
    while(good != 1 || y < 0 || y > 3000);

    return y;
}
```

```
int get_number()
/**
 *  @brief Получение номера из терминала
 */
{
    int n;
    int good;

    good = 1;
    do
    {
        printf("Enter number of track in album [1, 100]:
");
        good = scanf("%d", &n);
        clean_stdin();
        if (good != 1)
            puts(INPUT_ERROR);
        else if (n < 0)
            puts("\nSorry, but number should be great than
0");
        else if (n > 100)
            puts("\nSorry, but number should be less than
101");
    }
    while(good != 1 || n < 0 || n > 99);
```

```
    return n;
}



int get_genre()
/**
 *  @brief Получение номера жанра из терминала
 */
{
    int g;
    int good;

    good = 1;
    do
    {
        print_array(genres_array, GENRES_NUM);
        printf("Enter number of genre [0, %d]: ",
GENRES_NUM-1);
        good = scanf("%d", &g);
        clean_stdin();
        if (good != 1)
            puts(INPUT_ERROR);
        else if (g < 0)
            puts("\nSorry, but number should be great than
0");
        else if (g > GENRES_NUM-1)
```

```
            printf("\nSorry, but number should be less than
%d \n", GENRES_NUM);
    }
    while(good!= 1 || g < 0 || g > GENRES_NUM-1);

    return g;
}



void save_scanf(int* number)
{
    char good;
    do
    {
        good = scanf("%d", number);
        if (good != 1)
            puts(INPUT_ERROR);
        clean_stdin();
    }
    while(good != 1);
}
```

6.6. menu.h

```
#ifndef MENU_H
#define MENU_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "constants.h"
#include "help_func.h"
#include "dlist.h"

enum
{
    MAIN_MENU,
    INPUT_MENU,
    EDIT_MENU,
    SEARCH_MENU,
    OUTPUT_MENU,
    INPUT_TERM_MENU,
    EDIT_INFO_MENU,
    SORT_MENU1,
    SORT_MENU2,
    DELETE_MENU
};
```

```
int get_number( );
int get_year( );
int get_genre( );
char *get_string(unsigned char);
TRACK get_music_data( );
void delete_menu(LIST **);
void edit_info_menu(LIST *);
void search_menu(LIST *);
void sort_menu(LIST *);
void input_menu(LIST *);
void edit_menu(LIST **);
void output_menu(LIST *);
void print_menu(unsigned char, unsigned char *);
void help( );

#endif
```

6.7. menu.c

```
#include "menu.h"

TRACK get_music_data()
{
    TRACK music;

    system(CLEAR);
    music.artist = get_string(1);
    music.title = get_string(2);
    music.album = get_string(3);
    music.number = get_number();
    music.year = get_year();
    music.genre = get_genre();

    return music;
}



void print_menu(unsigned char menu, unsigned char *flags)
/*
    Вывод пунктов меню на экран
    ---
    @param char *flags - одномерный массив флагов, для
вывода
                         только доступных пунктов меню
```

```
*/
{
    switch (menu)
    {
        case MAIN_MENU:
            system(CLEAR);
            puts("Main menu");
            if (flags[1])
                puts("1. Help");
            if (flags[2])
                puts("2. Add music list");
            if (flags[3])
                puts("3. Edit music list");
            if (flags[4])
                puts("4. Search music");
            if (flags[5])
                puts("5. Print music list");
            if (flags[0])
                puts("0. Exit");
        break;
        case INPUT_MENU:
            system(CLEAR);
            puts("Input menu");
            puts("1. Input from term");
            puts("2. Input from file");
            puts("0. Return to Main Menu");
```

```
break;
case EDIT_MENU:
    system(CLEAR);
    puts("Edit menu");
    puts("1. Edit track info");
    puts("2. Sort music list");
    puts("3. Reverse list");
    puts("4. Random shuffle");
    puts("5. Swap");
    puts("6. Delete track");
    puts("0. Return to Main Menu");
break;
case SEARCH_MENU:
    system(CLEAR);
    puts("Choose field to search for");
    if (flags[0] == 1)
        puts("1. Artist");
    if (flags[1] == 1)
        puts("2. Title");
    if (flags[2] == 1)
        puts("3. Album");
    if (flags[3] == 1)
        puts("4. Number in album");
    if (flags[4] == 1)
        puts("5. Year");
    if (flags[5] == 1)
```

```
        puts("6. Genre");
break;
case OUTPUT_MENU:
    system(CLEAR);
    puts("Output menu");
    puts("1. Print all list");
    puts("2. Print one track");
    puts("3. Save list in file");
    puts("0. Return to Main Menu");
break;
case INPUT_TERM_MENU:
    system(CLEAR);
    if (flags[0] == 1)
    {
        puts("1. Add to head");
        puts("2. Add to tail");
        puts("3. Add after n-th element");
    }
break;
case EDIT_INFO_MENU:
    puts("\nEdit info menu");
    puts("1. Edit artist");
    puts("2. Edit title");
    puts("3. Edit album");
    puts("4. Edit number in album");
    puts("5. Edit year");
```

```
        puts("6. Edit genre");
        puts("0. Return to Edit Menu");
    break;
    case SORT_MENU1:
        system(CLEAR);
        puts("Sort menu");
        puts("1. Sort by artists");
        puts("2. Sort by title");
        puts("3. Sort by album");
        puts("4. Sort by number in album");
        puts("5. Sort by year");
        puts("0. Return to Edit Menu");
    break;
    case SORT_MENU2:
        system(CLEAR);
        puts("1. Increase");
        puts("2. Decrease");
    break;
    case DELETE_MENU:
        system(CLEAR);
        puts("Delete menu");
        puts("1. Delete track by number");
        puts("2. Delete all music list");
        puts("0. Return to Edit Menu");
    break;
}
```

```
    printf(">");
}


void input_menu(LIST *list)
{
    int variant,
        exit_flag;
    char answer;
    unsigned char flag;
    char *path = NULL;

    exit_flag = 1;
    do
    {

        print_menu(INPUT_MENU, NULL);
        save_scanf(&variant);

        switch (variant)
        {
            case 1:
                do
                {
                    if (is_empty(list))
                        flag = 0;
```

```
                    else
                        flag = 1;

                    if (flag != 0)
                    {
                        do
                        {
                            print_menu(INPUT_TERM_MENU,
&flag);
                            save_scanf(&variant);
                            if (variant < 1 || variant > 3)
                                print_msg(MENU_ERROR);
                        }
                        while (variant < 1 || variant > 3);
                    } else
                        variant = 1;

                    if (variant == 1)
                        push(list, get_music_data());
                    else if (variant == 2)
                        append(list, get_music_data());
                    else
                    {
                        if (flag == 1)
                        {
                            do
```

```
                            {
                                puts("Enter index of
element");
                                printf("Length of you list
is: %d\n", length(list));
                                printf(">");
                                save_scanf(&variant);
                                if (variant >=
length(list))
                                    puts(INDEX_ERROR);
                            }
                            while (variant >=
length(list));
                            if (variant == length(list)-1)
                                append(list,
get_music_data());
                            else
                                insert(list,
get_music_data(), variant);
                        }
                        else
                            print_msg(MENU_ERROR);
                    }

                    puts("\nDo you want to continue?
(y/n)");
                    printf(">");
                    scanf("%c", &answer);
                    clean_stdin();
```

```
    }
    while(answer == 'y' || answer == 'Y');
    if (list->head != NULL)
    {
        print_list(list);
        pause();
    }
break;
case 2:
    system(CLEAR);
    puts("Input filename (default: data.csv)");
    printf(">");
    path = get_string(0);
    get_list(list, path, ';');
    if (list->head != NULL)
    {
        print_list(list);
        pause();
    }
    free(path);
    path = NULL;
break;
case 0:
    exit_flag = 0;
break;
default:
```

```
                print_msg(MENU_ERROR);
            break;
        }

    }
    while(exit_flag);
}

void output_menu(LIST *list)
{
    int variant,
        exit_flag,
        n;
    char *path;

    exit_flag = 1;
    do
    {
        print_menu(OUTPUT_MENU, NULL);

        save_scanf(&variant);
        switch (variant)
        {
            case 1:
                print_list(list);
                pause();
```

```
            break;
            case 2:
                system(CLEAR);
                do
                {
                    puts("Input number of list element");
                    printf("Length of you list is: %d\n",
length(list));
                    printf(">");
                    save_scanf(&n);
                    if (n > length(list))
                        puts("Number can't be greater than
length");
                }
                while(n > length(list));
                print_list_element(get(list, n-1));
                pause();
            break;
            case 3:
                system(CLEAR);
                puts("Input filename");
                printf(">");
                path = get_string(0);
                save_list(list, path);
                free(path);
                path = NULL;
            break;
```

```
            case 0:
                exit_flag = 0;
            break;
            default:
                print_msg(MENU_ERROR);
            break;
        }

    }
    while(exit_flag);
}

void search_menu(LIST *list)
{
    int variant,
        i;

    LIST *search_list = NULL,
         *temp = NULL;

    TRACK search_param;
    char exit_flag;
    unsigned char flags[6];

    for (i = 0; i < 6; i++)
        flags[i] = 1;
```

```
    i = 0;
    search_list = list;
    do
    {
        print_menu(SEARCH_MENU, flags);

        save_scanf(&variant);
        switch (variant)
        {
            case 1:
                if (flags[0] == 1)
                {
                    search_param.artist = get_string(1);
                    flags[0] = 0;
                    temp = search_list;
                    search_list = search(search_list,
search_param, compare_artist);
                    if (i != 0)
                        soft_delete_list(&temp);
                    i++;
                    free(search_param.artist);
                }
                else
                    print_msg(MENU_ERROR);
            break;
```

```
            case 2:
                if (flags[1] == 1)
                {
                    search_param.title = get_string(2);
                    flags[1] = 0;
                    temp = search_list;
                    search_list = search(search_list,
search_param, compare_title);
                    if (i != 0)
                        soft_delete_list(&temp);
                    i++;
                    free(search_param.title);
                }
                else
                    print_msg(MENU_ERROR);
            break;
            case 3:
                if (flags[2] == 1)
                {
                    search_param.album = get_string(3);
                    flags[2] = 0;
                    temp = search_list;
                    search_list = search(search_list,
search_param, compare_album);
                    if (i != 0)
                        soft_delete_list(&temp);
                    i++;
```

```
                free(search_param.album);
            }
            else
                print_msg(MENU_ERROR);
        break;
        case 4:
            if (flags[3] == 1)
            {
                search_param.number = get_number();
                flags[3] = 0;
                temp = search_list;
                search_list = search(search_list,
search_param, compare_number);
                if (i != 0)
                    soft_delete_list(&temp);
                i++;
            }
            else
                print_msg(MENU_ERROR);
        break;
        case 5:
            if (flags[4] == 1)
            {
                search_param.year = get_year();
                flags[4] = 0;
                temp = search_list;
```

```
                    search_list = search(search_list,
search_param, compare_year);
                    if (i != 0)
                        soft_delete_list(&temp);
                    i++;
                }
                else
                    print_msg(MENU_ERROR);
            break;
            case 6:
                if (flags[5] == 1)
                {
                    search_param.genre = get_genre( );
                    flags[5] = 0;
                    temp = search_list;
                    search_list = search(search_list,
search_param, compare_genre);
                    if (i != 0)
                        soft_delete_list(&temp);
                    i++;
                }
                else
                    print_msg(MENU_ERROR);
            break;
            default:
                print_msg(MENU_ERROR);
            break;
```

```
        }

        if(i != 6)
        {
            puts("\nDo you want to choose more param?
(y/n)");
            printf(">");
            scanf("%c", &exit_flag);
            clean_stdin();
        }
        else
            exit_flag = 0;
    }
    while(exit_flag == 'Y' || exit_flag == 'y');

    if (!is_empty(search_list))
    {
        print_list(search_list);
        pause();
    }
    else
        print_msg(SEARCH_ERROR);

    if (i != 0)
        soft_delete_list(&search_list);
    if (i > 2)
```

```
        soft_delete_list(&temp);
}


void edit_menu(LIST **list)
{
    int variant,
        exit_flag,
        first,
        second;

    exit_flag = 1;
    do
    {
        print_menu(EDIT_MENU, NULL);

        save_scanf(&variant);
        switch (variant)
        {
            case 1:
                edit_info_menu(*list);
            break;
            case 2:
                sort_menu(*list);
            break;
            case 3:
```

```
                reverse(*list);
                print_list(*list);
                pause();
            break;
            case 4:
                shuffle(*list);
                print_list(*list);
                pause();
            break;
            case 5:
                do
                {
                    print_list(*list);
                    puts("\nInput first number");
                    printf(">");
                    save_scanf(&first);
                    puts("\nInput second number");
                    printf(">");
                    save_scanf(&second);
                    if (first > length(*list) || second >
length(*list)
                        || first < 1 || second < 1)
                        print_msg(INDEX_ERROR);
                } while (first > length(*list) || second >
length(*list) || first < 1 || second < 1);

                swap(*list, first-1, second-1);
```

```
                print_list(*list);
                pause();
            break;
            case 6:
                delete_menu(list);

                if (is_empty(*list))
                    exit_flag = 0;
            break;
            case 0:
                exit_flag = 0;
            break;
            default:
                print_msg(MENU_ERROR);
            break;
        }

    }
    while(exit_flag);
}

void edit_info_menu(LIST *list)
{
    int change_int,
        variant1,
        variant2,
```

```
        exit_flag;


    char *change_str;
    NODE *temp_node = NULL;
    exit_flag = 1;
    do
    {
        print_list(list);
        puts("\nEnter number of element of list (or 0 to
return to Edit Menu)");
        printf(">");
        save_scanf(&variant1);
        if (variant1 == 0)
           exit_flag = 0;
        else if (variant1 > length(list) || variant1 < 0)
            print_msg("Number should be greater than 0 and
less than length");
    }
    while(variant1 > length(list) || variant1 < 0);


    while (exit_flag)
    {
        temp_node = get(list, variant1-1);
        print_list_element(temp_node);
        print_menu(EDIT_INFO_MENU, NULL);
        save_scanf(&variant2);
```

```
switch (variant2)
{
    case 1:
        system(CLEAR);
        change_str = get_string(1);
        free(temp_node->data.artist);
        temp_node->data.artist = change_str;
    break;
    case 2:
        system(CLEAR);
        change_str = get_string(2);
        free(temp_node->data.title);
        temp_node->data.title = change_str;
    break;
    case 3:
        system(CLEAR);
        change_str = get_string(3);
        free(temp_node->data.album);
        temp_node->data.album = change_str;
    break;
    case 4:
        system(CLEAR);
        change_int = get_number( );
        temp_node->data.number = change_int;
    break;
    case 5:
```

```
                system(CLEAR);
                change_int = get_year();
                temp_node->data.year = change_int;
            break;
            case 6:
                system(CLEAR);
                change_int = get_genre();
                temp_node->data.genre = change_int;
            break;
            case 0:
                exit_flag = 0;
            break;
            default:
                print_msg(MENU_ERROR);
            break;
        }
    }
}

void sort_menu(LIST *list)
{
    int type;
    int rev;

    do
    {
```

```
        print_menu(SORT_MENU1, NULL);
        save_scanf(&type);

        if (type > 0 && type < 6)
        {
            do
            {
                print_menu(SORT_MENU2, NULL);
                save_scanf(&rev);
                if (rev > 2 || rev < 1)
                    print_msg(MENU_ERROR);
            }while(rev > 2 || rev < 1);

            sort(list, type);
            if (rev == 2)
                reverse(list);
            print_list(list);
            pause();
            type = 0;
        }
        else if(type != 0)
            print_msg(MENU_ERROR);
    }
    while (type != 0);
}
```

```
void delete_menu(LIST **list)
{
    int number,
        variant,
        exit_flag;

    exit_flag = 1;
    do
    {

        print_menu(DELETE_MENU, NULL);

        save_scanf(&variant);

        switch (variant)
        {
            case 1:
                print_list(*list);
                puts("\nEnter number of element of list");
                printf(">");
                save_scanf(&number);
                pop(*list, number-1);
                if (is_empty(*list))
                    exit_flag = 0;
                else
                {
```

```
                    print_list(*list);
                    pause();
                }
            break;
            case 2:
                delete_list(list);
                print_msg(LIST_DELETE);
                exit_flag = 0;
            break;
            case 0:
                exit_flag = 0;
            break;
            default:
                print_msg(MENU_ERROR);
            break;
        }

    }
    while(exit_flag);
}

void help()
{
    system(CLEAR);
    puts("This program will help you create your music
library.");
```

```
pause();
```

6.8. dlist.h

```
/*
    Библиотека для работы с двусвязным списком.

    ---
    Доступные операции:
    push - добавление в начало
    append - добавление в конец
    get - получение произвольного элемента
    insert - вставка в произвольную позицию
    pop - удаление элемента из произвольной позиции
    length - длина списка
    is_empty - проверка на пустоту списка
    swap - поменять местами два элемента
    sort - сортирока
    search - поиск
    reverse - разворот
    shuffle - рандомное перемешивание
*/

#ifndef DLIST_H
#define DLIST_H

// ------------------------- Структуры
------------------------

struct list_info
```

```
/*
    Информационная структура списка
*/
{
    char *artist; // Адрес начала строки,
    // которая содержит имя исполнителя
    char *title; // Адрес начала строки,
    // которая содержит название музыкальной композиции
    char *album; // Адрес начала строки,
    // которая содержит название альбома
    int genre; // Номер жанра музыки
    int year; // Год создания
    int number; // Номер в альбоме
};


struct list_node
/*
    Структура узла списка
*/
{
    struct list_info data; // данные карточки
    struct list_node *next; // указатель на следующий
элемент списка
    struct list_node *previous; // указатель на предыдущий
элемент списка
};
```

```
struct list
/*
    Структура списка
*/
{
    int size; // размер списка
    struct list_node *head; // указатель на первый элемент списка
    struct list_node *tail; // указатель на последний элемент списка
};

typedef struct list LIST;
typedef struct list_node NODE;
typedef struct list_info TRACK;

// ------------------------- Прототипы функций для списка -------------------------
char compare_artist(TRACK, TRACK);
char compare_title(TRACK, TRACK);
char compare_album(TRACK, TRACK);
char compare_number(TRACK, TRACK);
char compare_year(TRACK, TRACK);
char compare_genre(TRACK, TRACK);

LIST *create_list();
void memory_clear(TRACK *t);
```

```
void delete_list(LIST **);
void soft_delete_list(LIST **);
void push(LIST* list, TRACK data);
void append(LIST *, TRACK);
NODE *get(LIST *, unsigned);
void insert(LIST *, TRACK, unsigned);
void pop(LIST *, unsigned);
void swap(LIST *, unsigned, unsigned);
char sort_compare(NODE *, NODE *, unsigned);
void sort(LIST *, int);
LIST *search(LIST*, TRACK, char(*compare)(TRACK, TRACK));
void reverse (LIST *);
void shuffle (LIST *);
int length(LIST *);
char is_empty(LIST *);
void print_list(LIST *);
void print_list_element(NODE *);
void save_list(LIST *, char *);
TRACK fill_node(char **);
char **simple_split(char *, int, char);
void get_list(LIST*, char*, char);

#endif
```

6.9. dlist.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "dlist.h"
#include "constants.h"
#include "help_func.h"

LIST *create_list()
/**
 *   @brief Создание двусвязного списка
 */
{
    LIST *temp = NULL;

    temp = malloc(sizeof(LIST));

    if (temp != NULL)
    {
        temp->size = 0;
        temp->head = NULL;
        temp->tail = NULL;
    }
    return temp;
}
```

```
void delete_list(LIST **list)
/**
 *    @brief Удаление двусвязного списка
 */
{
    NODE *temp_node = NULL;
    NODE *previous = NULL;

    if (*list != NULL)
    {
        temp_node = (*list)->head;
        while (temp_node)
        {
            previous = temp_node;
            memory_clear(&(previous->data));
            temp_node = temp_node->next;
            free(previous);
        }
        free(*list);
        (*list) = NULL;
    }
}

void soft_delete_list(LIST **list)
/**
```

```
 *  @brief Удаление двусвязного списка
 */
{
    NODE *temp_node = NULL;
    NODE *previous = NULL;

    if (*list != NULL)
    {
        temp_node = (*list)->head;
        while (temp_node)
        {
            previous = temp_node;
            temp_node = temp_node->next;
            free(previous);
        }
        free(*list);
        (*list) = NULL;
    }
}

void push(LIST *list, TRACK data)
/**
 *  @brief Добаление нового элемента в начало списка.
Сложность O(1)
 *  ---
 *  @param LIST *list - указатель на список
 *  @param TRACK data - данные нового элемента
```

```
 */
{
    NODE *new_node = NULL;

    new_node = malloc(sizeof(NODE));
    if (new_node != NULL)
    {
        new_node->data = data;
        new_node->next = list->head;
        new_node->previous = NULL;
        if (list->head)
            list->head->previous = new_node;
        list->head = new_node;

        if (list->tail == NULL)
            list->tail = new_node;

        list->size++;
    }
}



void pop(LIST *list, unsigned index)
/**
 *  @brief Удаление элемента из списка. Сложность O(n)
 *  ---
 *  @param LIST *list - список.
```

```
 *  @param unsigned index - номер элемента
*/
{
    NODE *node = NULL;
    node = get(list, index);
    if (node != NULL)
    {
        if (node->previous)
            node->previous->next = node->next;

        if (node->next)
            node->next->previous = node->previous;

        memory_clear(&(node->data));

        if (!node->previous)
            list->head = node->next;

        if (!node->next) {
            list->tail = node->previous;
        }

        free(node);

        list->size--;
    }
}
```

```
void append(LIST *list, TRACK data)
/**
 *  @brief Добавление нового элемента в конец списка.
Сложность O(1)
 *  ---
 *  @param LIST *list - указетель на список.
 *  @param TRACK data - данные нового элемента.
 */
{
    NODE *new_node = NULL;

    new_node = malloc(sizeof(NODE));

    if (new_node != NULL)
    {
        new_node->data = data;
        new_node->next = NULL;
        new_node->previous = list->tail;
        if (list->tail)
            list->tail->next = new_node;
        list->tail = new_node;

        if (list->head == NULL)
            list->head = new_node;
        list->size++;
    }
```

```
}


void insert(LIST *list, TRACK data, unsigned index)
/**
 *  @brief Добавление нового элемента после определенного
элемента списка. Сложность O(n)
 *  ---
 *  @param LIST *list - указетель на список
 *  @param TRACK data - данные нового элемента
 *  @param unsigned index - позиция элемента
 */
{
    NODE *elm = NULL;
    NODE *new_node = NULL;
    elm = get(list, index);
    if (elm != NULL)
    {
        new_node = malloc(sizeof(NODE));
        new_node->data = data;
        new_node->previous = elm;
        new_node->next = elm->next;

        if (elm->next)
            elm->next->previous = new_node;

        elm->next = new_node;
```

```
        if (elm->previous == NULL)
            list->head = elm;

        if (elm->next == NULL)
            list->tail = elm;

        list->size++;
    }
}



NODE *get(LIST *list, unsigned index)
/**
 *  @brief Получение элемента списка. Сложность O(n/2) =
O(n).
 *  ---
 *  @param LIST *list - указатель на список
 *  @param unsigned index - позиция элемента
 *  @return NODE - нужный элемент списка
 */
{
    NODE *temp_node = NULL;
    int i;

    // если элемент в первой половине списка,
    // то идем с начала списка
```

```
    if (index < list->size/2)
        for (i = 0, temp_node = list->head;
            temp_node && i < index;
            temp_node = temp_node->next, i++);
    else
    // если элемент во второй половине списка,
    // то идем с конца списка
        for (i = list->size - 1, temp_node = list->tail;
             temp_node && i > index;
             temp_node = temp_node->previous, i--);

    return temp_node;
}


void swap(LIST *list, unsigned pos1, unsigned pos2)
/**
 *  @brief Поменять местами два элемента. Сложность O(n)
 *  ---
 *  @param LIST *list - указатель на список
 *  @param unsigned pos1 - позиция первого элемента
 *  @param unsigned pos2 - позиция второго элемента
 */
{
    TRACK tmp;
    NODE *first = NULL,
```

```
         *last = NULL;

    first = list->head;
    last = list->head;
    while(pos1--)
        first = first->next;

    if(first == NULL)
        return;

    while(pos2--)
        last = last->next;

    if(last == NULL)
        return;

    tmp = first->data;
    first->data = last->data;
    last->data = tmp;
}



LIST *search(LIST *list, TRACK search_param, char(*compare)
(TRACK, TRACK))
/**
 *  @brief Поиск по элементам списка
 *  ---
```

```
 *  @param LIST *list - список
 *  @param TRACK search_param - параметры поиска по списку
 *  @param char*()(TRACK, TRACK) - указатель на функцию
сравнения
 *  @return LIST* - список
 */
{
    LIST* search_result = NULL;
    NODE* temp_node = NULL;

    search_result = create_list();
    if (search_result != NULL)
        for (temp_node = list->head; temp_node; temp_node =
temp_node->next)
            if (compare(temp_node->data, search_param))
                append(search_result, temp_node->data);

    return search_result;
}



void sort(LIST *head, int type)
/**
 *  @brief Сортировка элементов списка методом пузырька
 *  ---
 *  @param LIST *list - список
 *  @param int type - тип сортируемого значения
```

```
 *  @return NODE - отсортированный список
 */
{
    int i, j, n;

    n = length(head);
    for (i = n - 1; i > 0; i--)
        for (j = 0; j < i; j++)
            if (sort_compare(get(head, j), get(head, j+1),
type))
                swap(head, j, j+1);
}



char sort_compare(NODE *first, NODE *second, unsigned type)
{
    char res;

    if ((type == 1 && strcmp((first->data).artist, (second-
>data).artist) > 0)
        ||(type == 2 && strcmp((first->data).title,
(second->data).title) > 0)
        ||(type == 3 && strcmp((first->data).album,
(second->data).album) > 0)
        ||(type == 4 && (first->data).number > (second-
>data).number)
        ||(type == 5 && (first->data).year > (second-
>data).year)
        )
```

```
        res = 1;
    else
        res = 0;

    return res;
}


void reverse(LIST *list)
/**
 *  @brief Реверс двусвязного списка. Сложность O(n)
 *  ---
 *  @param LIST *list - указатель на список
 */
{
    NODE *left = NULL,
         *right = NULL;
    TRACK temp;

    left = list->head;
    right = list->tail;

    while (left != right && left->previous != right)
    {
        temp = left->data;
        left->data = right->data;
        right->data = temp;
```

```
        left = left->next;
        right = right->previous;
    }
}

void shuffle(LIST *list)
/**
 *  @brief Случайное перемешивание двусвязного списка.
 *  ---
 *  @param LIST *list - указатель на список
 */
{
    int i, j;

    srand(time(NULL));
    for (i = length(list) - 1; i >= 1; i--)
    {
        j = rand() % (i+1);
        swap(list, i, j);
    }
}

int length(LIST *list)
/**
 *  @brief Получение длины списка. Сложность O(1)
```

```
 *  ---
 *  @param LIST *list - указатель на список
 *  @return int - длина списка
 */
{
    return list->size;
}

char is_empty(LIST *list)
/**
 *  @brief Проверка списка на пустоту. Сложность O(1)
 *  ---
 *  @param LIST *list - указатель на список
 *  @return char - 1 - если пустой, 0 - иначе
 */
{
    return (list == NULL || list->head == NULL);
}

void print_list(LIST *list)
/**
 *  @brief Вывод списка в виде таблицы
 *  ---
 *  @param LIST *list - указатель на список
 */
{
```

```
    unsigned count;
    NODE *temp_node = NULL;

    system(CLEAR);
    print_header();

    temp_node = list->head;
    count = 1;
    while(temp_node)
    {
        printf("|%6u |%21s |%21s |%21s |%7d |%5d |%10s |\
n",
            count, (temp_node->data).artist,
            (temp_node->data).title, (temp_node-
>data).album,
            (temp_node->data).number,(temp_node-
>data).year,
            genres_array[(temp_node->data).genre]);

        temp_node = temp_node->next;
        count++;
    }
}



void print_list_element(NODE *node)
/**
 *  @brief Вывод элемента списка в виде таблицы
```

```
 *  ---
 *  @param LIST *list - указатель на список
 */
{
    system(CLEAR);
    print_header();

    printf("|%6d |%21s |%21s |%21s |%7d |%5d |%10s |\n",
        1, (node->data).artist, (node->data).title, (node-
>data).album,
        (node->data).number,(node->data).year,
genres_array[(node->data).genre]);
}

void save_list(LIST *list, char *filename)
/**
 *  @brief Сохраниение списка в csv файл
 *  ---
 *  @param LIST *list - указатель на список
 *  @param char *filename - путь до файла
 */
{
    FILE *df;
    NODE *temp_node = NULL;

    df = fopen(filename, "w");
    if(df != NULL)
```

```
    {
        for (temp_node = list->head; temp_node; temp_node =
temp_node->next)
            fprintf(df, "%s;%s;%s;%d;%d;%d\n",temp_node-
>data.artist, temp_node->data.title,
            temp_node->data.album, temp_node->data.number,
            temp_node->data.year, temp_node->data.genre);


        fclose(df);
        print_msg(FILE_SUCCESS);
    }
    else
        print_msg(FILE_ERROR);
}


void get_list(LIST *list, char* filename, char separator)
/**
 *  @brief Получение списка из csv файла
 *  ---
 *  @param LIST *list - указатель на список
 *  @param char *filename - путь до файла
 *  @param char separator - разделитель
 */
{
    int slen,
        flag=1;
    char s1[MAXSTR];
```

```
char **s2 = NULL;
FILE *df;

df = fopen(filename, "r");
if(df != NULL)
{
    while(fgets(s1,MAXSTR,df) != NULL && flag)
    {
        slen = strlen(s1);
        s1[slen-1] = '\0';
        slen = slen - 1;

        s2 = simple_split(s1,slen,separator);
        if(s2 != NULL)
        {
            append(list, fill_node(s2));
            clear_str_array(s2,6);
            free(s2);
        }
        else
        {
            flag=0;
            puts("Row data not available!");
        }
    }
    fclose(df);
```

```
    }
    else
        print_msg(FILE_ERROR);

}


TRACK fill_node(char **s2)
/**
 *  @brief Заполнение структуры
 *  ---
 *  @param char **s2 - массив строк
 *  @return TRACK - заполненная структура
 */
{
    TRACK p;
    int len1, len2, len3;

    len1=strlen(s2[0]);
    len2=strlen(s2[1]);
    len3=strlen(s2[2]);

    p.artist = malloc((len1+1)*sizeof(char));
    p.title = malloc((len2+1)*sizeof(char));
    p.album = malloc((len3+1)*sizeof(char));

    if((p.artist!=NULL)&&(p.title!=NULL)&&(p.album!=NULL))
```

```
    {
        strcpy(p.artist, s2[0]);
        strcpy(p.title, s2[1]);
        strcpy(p.album, s2[2]);
        p.number = strtol(s2[3], NULL, 10);
        p.year = strtol(s2[4], NULL, 10);
        p.genre = strtol(s2[5], NULL, 10);
    }
    else
        puts("Out of memory! Program terminated");

    return p;
}


char **simple_split(char *str, int length, char sep)
/**
 *  @brief Разбиение строки на массив строк
 *  ---
 *  @param char *str - строка для разбиения
 *  @param int length - длина строки
 *  @param char sep - разделитель
 *  @return char** - массив строк
 */
{
    char **str_array = NULL;
```

```
int i,j,k,m,key,count;
for(j=0, m=0; j < length; j++)
    if(str[j]==sep) m++;

key=0;
str_array = calloc(m+1, sizeof(char*));
if(str_array != NULL)
{
    for (i = 0; i <= m; i++)
        str_array[i] = NULL;
    for(i=0,count=0;i<=m;i++,count++)
    {
        str_array[i] = calloc(length, sizeof(char));
        if(str_array[i]!=NULL)
            key=1;
        else
        {
            key=0;
            i=m;
        }
    }
    if(key)
    {
        k=0;
        m=0;
        for(j=0;j<length;j++)
```

```
            {
                if(str[j]!=sep)
                    str_array[m][j-k] = str[j];
                else
                {
                    str_array[m][j-k] = '\0';
                    k = j+1;
                    m++;
                }
            }
            str_array[m][j-k] = '\0';
        }
        else
        {
            clear_str_array(str_array, count);
            free(str_array);
        }
     }
     return str_array;
}

void memory_clear(TRACK *t)
/**
 *  @brief Освобождение памяти из-под полей структуры
 *  ---
 *  @param TRACK* t - указатель на структуру, у которой
очищаются поля
```

```
 */
{
    if(t->artist != NULL)
        free(t->artist);
    if(t->title != NULL)
        free(t->title);
    if(t->album != NULL)
        free(t->album);

    t->artist = NULL;
    t->title = NULL;
    t->album = NULL;
}

char compare_artist(TRACK first, TRACK second)
{
    return strcmp(first.artist, second.artist) == 0;
}

char compare_title(TRACK first, TRACK second)
{
    return strcmp(first.title, second.title) == 0;
}

char compare_album(TRACK first, TRACK second)
{
```

```
    return strcmp(first.album, second.album) == 0;
}

char compare_number(TRACK first, TRACK second)
{
    return first.number == second.number;
}

char compare_year(TRACK first, TRACK second)
{
    return first.year == second.year;
}

char compare_genre(TRACK first, TRACK second)
{
    return first.genre == second.genre;
}
```

Заключение

При выполнении курсовой работы были получены практические навыки поэтапного решение содержательной задачи, связанной с обработкой структур данных.