

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 на тему
“Разработка электронной картотеки” по
дисциплине “Программирование”**

Выполнил студент гр. 9308

Хамитов А.К

Проверил к.т.н., доцент

Перязева Ю.В

Санкт-Петербург, 2020

Оглавление

Цель	3
1. Задание	3
2. Уточнение задания.....	3
3. Контрольные примеры	Ошибка! Закладка не определена.
4. Краткое описание алгоритма	10
5. Функции	11
5.1 Главная функция	11
5.2 Функция dbl_list.....	12
5.3 Функция add_item.....	13
5.4 Функция get_string	15
5.5 Функция get_int.....	16
5.6 Функция get_float.....	17
5.7 Функция get_category	18
5.8 Функция fill_node	19
5.9 Функция make_head.....	20
5.10 Функция print_managers	21
5.11 Функция selected	22
5.12 Функция add_first.....	23
5.13 Функция add_last.....	24
5.14 Функция insert	25
5.15 Функция swap.....	26
5.16 Функция remove_node	27
5.17 Функция copy_node.....	28
5.18 Функция create_node	29
5.19 Функция Menu.....	30
5.20 Функция get_database.....	31
5.21 Функция write_to_file.....	32
6. Текст программы	36
6.1 main.c	36
6.2 dbl_list.c	37
6.3 get.c	43
6.4 w_file.c	44
6.5 common.c	47
6.6 common.h	52
6.7 dbl_list.h.....	53
6.8 get.h.....	55
6.9 w_file.h.....	55
7. Пример работы программы	56
Заключение	57

Цель

Целью курсовой работы является законченное поэтапное решение содержательной задачи, связанной с реализацией структур, линейных двусвязных списков на языке программирования C/C++.

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. Уточнение задания

Выбранная предметная область – записи доходов и затрат.

Исходя из выбранной предметной области, были выбраны следующие поля структуры:

Таблица 1. Описание полей структуры manager

Имя поля	Тип	Назначение
expenses_income	int	Доходы или затраты
category	char*	Категория затрат/доходов
description	char*	Описание записи
money	float	Количество денег

Поля вспомогательной структуры “узла”:

Таблица 2. Описание полей структуры node

Имя поля	Тип	Назначение
info	manager	Информация об узле
next	Struct manager_elem*	Указатель на адрес следующего узла

Поля вспомогательной структуры “головы”:

Таблица 3. Описание полей структуры Head

Имя поля	Тип	Назначение
count	int	Количество узлов в списке
first	Node *	Указатель на первый элемент в списке
last	Node *	Указатель на последний элемент в списке

Поле expenses_income определяется бинарно: 1 – доходы, 2 – затраты. Поле description должно содержать строку не длиннее 25. Поле money должно содержать положительное число.

Поле category определяется введенным числом пользователя от 1-5. Сопоставление числа и категории представлены ниже:

- 1 - Питание
- 2 – Транспорт
- 3 – Одежда
- 4 – Социальное

5 – Подарок

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
Common.c	Содержит функции menu, help, dbl_list, dbl_cycle_list
Dbl_lst.c	Содержит функции make_head, print_managers, get_category, fill_node, create_node, add_first, add_last, insert, add_item, copy_node, swap, clean_node, remove_node, compare, sort, selected, clean_list, edit_node, search, search_managers
Get.c	Содержит функции get_string, get_int, get_float
W_file.c	Содержит функции clear_str_Array, convert_to_node, simple_split, get_database, cycle_get_database, write_to_file
Common.h	Содержит прототипы функций и предварительные объявления для common.c
Const.h	Константы для линейного двусвязного списка
Dbl_list.h	Содержит прототипы функций и предварительные объявления для dbl_list.c
W_file.h	Содержит прототипы функций и предварительные объявления для w_file.c

Пользователю выводится меню с каким списком работать: линейным или кольцевым двусвязным.

1. Работа с линейным двусвязным списком
2. Работа с кольцевым двусвязным списком
3. Выход

1 и 2 пункты выведут меню программы с подобной иерархией:

1 - Ввести

1.1 Взять с файла

- 1.1.1 Взять с файла по умолчанию?
- 1.1.2 Выбрать файл
- 1.1.3 Вернуться назад

1.2 Ввести с консоли

- 1.2.1 Добавление в начало
 - 1.2.1.1 Выбор дохода или затраты
 - 1.2.1.2 Ввод описания
 - 1.2.1.3 Ввод количества денег
 - 1.2.1.4 Продолжить ввод или нет

1.2.2 Добавление в конец

- 1.2.2.1 Выбор дохода или затраты
- 1.2.2.2 Ввод описания
- 1.2.2.3 Ввод количества денег
- 1.2.2.4 Продолжить ввод или нет

1.2.3 Добавление в n-ую позицию

- 1.2.3.1 Выбор дохода или затраты
- 1.2.3.2 Ввод описания
- 1.2.3.3 Ввод количества денег
- 1.2.3.4 Вставка в n-ую позицию
- 1.2.3.5 Продолжить ввод или нет

2 - Контрольный вывод

3 - Действия со списком

3.1 Перестановка двух элементов

- 3.1.1 Ввод номера первой позиции
- 3.1.2 Ввод номера второй позиции
- 3.1.3 Еще одна перестановка? Иначе назад

3.2 Удаление элемента

- 3.2.1 Хотите увидеть список заметок?
- 3.2.2 Удалить элемент с номером [от 1 до n]

- 3.2.3 Удалить еще один элемент? Иначе назад
 - 3.3 Сортировка
 - 3.3.1 1 - по цене, 2 – по доходам/затратам
 - 3.3.1.1 По возрастанию или убыванию?
 - 3.4 Добавление элемента
 - 3.4.1 Добавление в начало
 - 3.4.1.1 Выбор дохода или затраты
 - 3.4.1.2 Ввод описания
 - 3.4.1.3 Ввод количества денег
 - 3.4.1.4 Продолжить ввод или нет
 - 3.4.2 Добавление в конец
 - 3.4.2.1 Выбор дохода или затраты
 - 3.4.2.2 Ввод описания
 - 3.4.2.3 Ввод количества денег
 - 3.4.2.4 Продолжить ввод или нет
 - 1.1.1 Добавление в n-ую позицию
 - 3.4.3.1 Выбор дохода или затраты
 - 3.4.3.2 Ввод описания
 - 3.4.3.3 Ввод количества денег
 - 3.4.3.4 Вставка в n-ую позицию
 - 3.4.3.5 Продолжить ввод или нет
 - 3.5 Сохранение изменений в файл
- 4 - Фильтр
- 4.1 Фильтр по 3 критериям
- 5 - Вывод результата
- 5.1 Вывод в файл
 - 5.2 Вывод в консоль
 - 5.1 Назад
- 6 - Справка
- 6.1 Назад
- 0 – Выход

При выборе пункта 1 пользователю даются на выбор готовые данные из файла *database.txt* или возможность указать путь до другого файла, либо ввести данные через терминал, в этом случае программа должна получить на вход значения полей первого элемента списка, затем пользователю предлагаются добавить еще один в начало, середину или конец списка. Поля структур вводятся до тех пор, пока пользователь не захочет прекратить ввод. Таким образом формируется либо линейный, либо кольцевой двусвязный список.

При выборе пункта 2 происходит вывод введенных пользователем полей структур.

При выборе пункта 3 пользователю даются варианты действий со списками: перестановка двух элементов, удаление элемента, сохранение изменений исходного списка в файл, редактирование элемента списка, поиск элементов по критериям.

При выборе пункта 4 формируется новый список, в который входят структуры из первоначального списка, удовлетворяющие условиям, которые введет пользователь:

1. Принадлежат доходам или затратам
2. Принадлежат выбранной категории
3. Структуры, суммы которых меньше указанной пользователем

При выборе пункта 5 пользователю дается на выбор загрузить новый список в файл или вывести в терминал. Если не было введено структур, удовлетворяющим трем условиям, выводится соответствующее сообщение.

При выборе пункта 6 пользователю выводится краткая справка о работе программы

При выборе 0 пункта происходит выход программы.

3. Контрольные примеры

Контрольные примеры приведены в таблице 5.

Таблица 5. Контрольные примеры

№	Исходные данные				Результаты		
	Доход затраты	Категория (category)	Описание (description)	Кол-во денег (money)	Критерии		
					По затратам	По категории - транспорт	По максимальной цене - 500
1	Затраты	Транспорт	Утреннее метро	55	Затраты. Транспорт. 500		
2	Затраты	Одежда	Любимая футболка	-100	Число должно быть положительным		
3	Доход	Социальное	Подарок на день рождения	600	Заметок не найдено		
4	Доход	Другое	Выигрыш в лотерею	100000	Заметок не найдено		
5	Затраты	Транспорт	Трамвай	25	Затраты. Транспорт. 25		
6	Затраты	Питание	Учпучмак	85	Заметок не найдено		
7	Доход	Другое	Денежная премия	1000	Заметок не найдено		
8	Затраты	Питание	Сосиска в тесте	75	Заметок не найдено		

4. Краткое описание алгоритма

Начало программы.

Шаг №1. Вывод меню (функция *Menu*).

Шаг №2. Выбор пользователем пункта меню.

Шаг №3. Переход к пункту, выбранным пользователем:

Пункт 1-ый: Ввод

Пункт 2-ый: Контрольный вывод

Пункт 3-ый: Действия со списком

Пункт 4-ый: Фильтр

Пункт 5-ый: Вывести результат

Пункт 6-ый: Вывод справки для пользователя

Пункт 0-ой: Конец программы

Шаг №5: Если пользователь не захотел выйти, то переход к шагу 1.

Шаг №6. Иначе, конец программы.

5. Функции

5.1 Главная функция

Назначение:

Является точкой входа в программу.

Прототип:

```
int main(void)
```

Пример вызова:

```
Main();
```

Описание переменных:

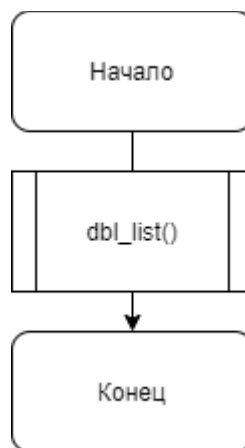
Описание переменных приведено в таблице 6.

Таблица 6. Описание переменных главной функции

Имя переменной	Тип	Назначение
Переменные отсутствуют		

Схема алгоритма представлена на рисунке 1.

Рисунок 1. Схема алгоритма главной функции



5.2 Функция dbl_list

Назначение:

Функция для работы с двусвязными линейными списками.

Прототип:

```
void dbl_list(void)
```

Пример вызова:

```
Dbl_list();
```

Описание переменных:

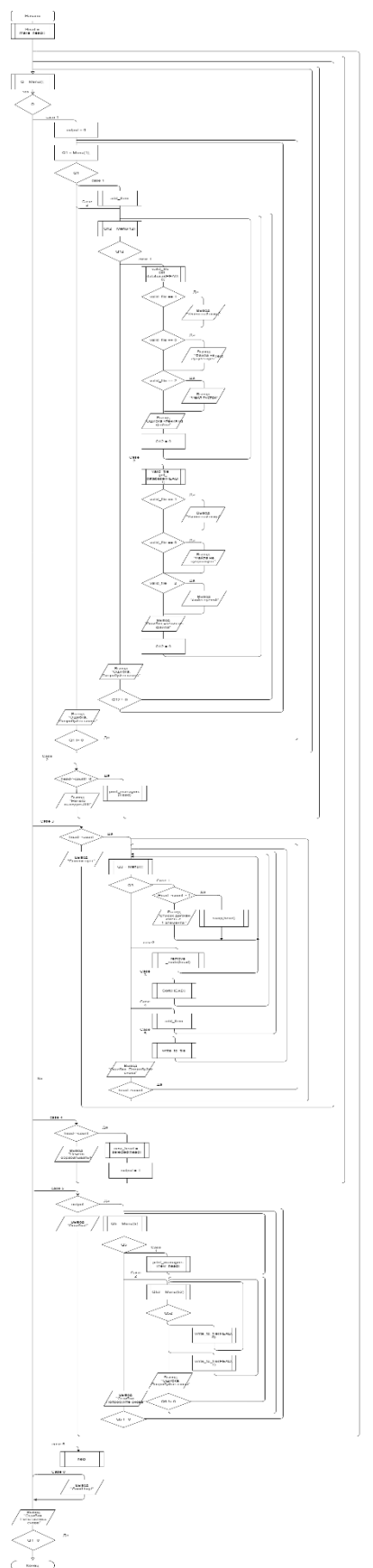
Описание переменных приведено в таблице 7.

Таблица 7. Описание переменных функции

Имя переменной	Тип	Назначение
Q	Int	Выбор пункта меню
Q3	int	Выбор пункта подменю
first	int	Первый элемент swap
second	int	Второй элемент swap
HEAD	Head*	Указатель на “голову списка”
NEW_HEAD	Head*	Указатель на “голову результирующего списка”
p	Node*	Указатель на адрес очередного узла

Схема алгоритма представлена на рисунке 2.

Рисунок 2. Схема алгоритма функции `dbl_list`



5.3 Функция add_item

Назначение:

Функция для выбора пользователя: куда добавить очередной элемент списка

Прототип:

```
Void add_item(Head*, int*)
```

Пример вызова:

```
add_item(head, &bl)
```

Описание переменных:

Описание переменных приведено в таблице 8.

Таблица 8. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	HEAD	Head*	Указатель на голову
Формальная	bl	Int*	Проверка на выделение динамической памяти
Локальная	p	Node*	Указатель на очередной элемент списка
Локальная	c	char	Выбор пользователя: добавить элемент в начало, середину, конец списка

5.4 Функция `get_string`

Назначение:

Функция чтения строки

Прототип:

```
char *get_string();
```

Пример вызова:

```
list->description = get_string();
```

Описание переменных:

Описание переменных приведено в таблице 9.

Таблица 9. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	c	char	Ввод символа
Локальная	string	Char*	Указатель на вектор символов(строка)
Локальная	i	int	Переменная для отслеживания количества символов

5.5 Функция get_int

Назначение:

Функция чтения числа типа int.

Прототип:

```
int get_int(void)
```

Пример вызова:

```
List->money = get_int();
```

Описание переменных:

Описание переменных приведено в таблице 10.

Таблица 10. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	Line	char	Буфер
Локальная	curChar	Char	Для записи Enter
Локальная	Temp	Int	Приминает 1 при успешной записи числа, иначе 0
Локальная	result	Int	Возвращаемое число

5.6 Функция get_float

Назначение:

Функция для чтения числа типа float.

Прототип:

```
int get_float(void)
```

Пример вызова:

```
List->money = get_float();
```

Описание переменных:

Описание переменных приведено в таблице 11.

Таблица 11. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	Line	char	Буфер
Локальная	curChar	Char	Для записи Enter
Локальная	Temp	Int	Приминает 1 при успешной записи числа, иначе 0
Локальная	result	Int	Возвращаемое число

5.7 Функция get_category

Назначение:

Функция для выбора категории пользователем.

Прототип:

Char *get_category(void)

Пример вызова:

List->category = get_category();

Описание переменных:

Описание переменных приведено в таблице 12.

Таблица 12. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	choice	Char*	Указатель на выбранную категорию
Локальная	C	Char	Проверка на конец файла
Локальная	q	I	Переменная, содержащая число – выбор пользователя категории

5.8 Функция fill_node

Назначение:

Функция для добавления записи.

Прототип:

Void *fill_node(manager, int*)

Пример вызова:

Fill_node(list, &bl);

Описание переменных:

Описание переменных приведено в таблице 13.

Таблица 13. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	manager*	Указатель на новую структуру
Формальная	bl	Int*	Проверка на выделение динамической памяти

5.9 Функция make_head

Назначение:

Функция создание головы

Прототип:

```
Head *make_head(int*);
```

Пример вызова:

```
Head = make_head(&bl);
```

Описание переменных:

Описание переменных приведено в таблице 14.

Таблица 14. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	ph	Head	Указатель на “голову”
Локальная	bl	Int*	Указатель на проверку выделения динамической памяти

5.10 Функция print_managers

Назначение:

Выводит входные данные(записи).

Прототип:

Void print_managers(manager, int)

Пример вызова:

Print_managers(data, count);

Описание переменных:

Описание переменных приведено в таблице 15.

Таблица 15. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	list	manager	Указатель на вектор структур
Формальная	count	Int	Количество записей
Локальная	i	int	Переменная для цикла For

5.11 Функция selected

Назначение:

Функция для обработки записей по критериям.

Прототип:

Head *selected(Head *, int*)

Пример вызова:

New_head = selected(my_head, &bl);

Описание переменных:

Описание переменных приведено в таблице 16.

Таблица 16. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	New_head	Head*	Указатель на новый вектор структур
Локальная	Node	Node*	Указатель на адрес узла в списке
Локальная	i	int	Переменная в цикле
Локальная	Expenses_income	int	Критерий: выбор пользователя доходы или затраты
	Max_money	float	Критерий: максимальная сумма
Локальная	category	Char*	Указатель на адрес строки категории
Формальная	My_head	Head*	Указатель на адрес "голову" списка
Локальная	bl	Int*	Указатель на проверку выделения динамической памяти

5.12 Функция add_first

Назначение:

Функция для добавления элемента в начало списка.

Прототип:

```
Void add_first(Head*, Node*)
```

Пример вызова:

```
Add_first(my_head, new_node);
```

Описание переменных:

Описание переменных приведено в таблице 17.

Таблица 17. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	My_head	Head*	Указатель на адрес “голову” списка
Формальная	New_node	Node*	Указатель на новый “узел”

5.13 Функция add_last

Назначение:

Функция для добавления элемента в конец списка.

Прототип:

Void add_last(Head*, Node*)

Пример вызова:

Add_last(my_head, new_node);

Описание переменных:

Описание переменных приведено в таблице 18.

Таблица 18. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	My_head	Head*	Указатель на адрес “голову” списка
Формальная	New_node	Node*	Указатель на новый “узел”

5.14 Функция insert

Назначение:

Функция для добавления элемента в любую позицию списка.

Прототип:

```
Void insert(Head*, Node*)
```

Пример вызова:

```
insert(my_head, *new_node);
```

Описание переменных:

Описание переменных приведено в таблице 19.

Таблица 19. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	My_head	Head*	Указатель на адрес “голову” списка
Формальная	New_node	Node*	Указатель на новый “узел”
Локальная	i	int	Переменная в цикле
Локальная	pos	int	Номер позиции в списке
Локальная	p	Node*	Указатель на “узел” для работы со списком

5.15 Функция swap

Назначение:

Функция для перестановки элементов.

Прототип:

Void swap(Head*)

Пример вызова:

swap(HEAD);

Описание переменных:

Описание переменных приведено в таблице 20.

Таблица 20. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	HEAD	Head*	Указатель на адрес “голову” списка
Локальная	First	int	Номер первой позиции в списке
Локальная	Second	int	Номер второй позиции в списке
Локальная	P_one	Node*	Указатель на “узел”, требуемый для перестановки двух элементов в списке
Локальная	P_two	Node*	Указатель на “узел”, требуемый для перестановки двух элементов в списке
Локальная	Buff_one	Node*	Указатель на “узел”, требуемый для перестановки двух элементов в списке
Локальная	Buff_two	Node*	Указатель на “узел”, требуемый для перестановки двух элементов в списке

5.16 Функция remove_node

Назначение:

Функция для удаления элементов в списке

Прототип:

```
Void remove_node(Head*)
```

Пример вызова:

```
remove_node (my_head);
```

Описание переменных:

Описание переменных приведено в таблице 21.

Таблица 21. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	My_head	Head*	Указатель на адрес “голову” списка
Локальная	i	int	Переменная в цикле
Локальная	pos	int	Номер позиции в списке
Локальная	p	Node*	Указатель на “узел”, требуемые для удаления элемента в списке
Локальная	buff	Node*	Указатель на “узел”, требуемые для удаления элемента в списке

5.17 Функция `copy_node`

Назначение:

Функция для удаления элементов в списке

Прототип:

```
Void copy_node(Node*, int*)
```

Пример вызова:

```
p = copy_node (node, &bl);
```

Описание переменных:

Описание переменных приведено в таблице 22.

Таблица 22. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	NODE	Node*	Указатель на адрес элемент списка
Локальная	i	int	Переменная в цикле
Локальная	p	Node*	Указатель на “узел”, требуемые для удаления элемента в списке
Локальная	bl	Int*	Указатель на проверку выделения динамической памяти

5.18 Функция create_node

Назначение:

Функция для создания элемента списка

Прототип:

Node create_node(Node*, int*)

Пример вызова:

p = create_node (node, &bl);

Описание переменных:

Описание переменных приведено в таблице 23.

Таблица 23. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	New_node	Node*	Указатель на адрес элемент списка
Локальная	bl	Int*	Указатель на проверку выделения динамической памяти

5.19 Функция Menu

Назначение:

Функция вывода меню.

Прототип:

Int Menu();

Пример вызова:

Menu();

Описание переменных:

Описание переменных приведено в таблице 24.

Таблица 24. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Локальная	Q	int	Возвращаемое число, содержащее пункт меню, выбираемое пользователем

5.20 Функция get_database

Назначение:

Функция для чтения с файла (линейного двусвязного списка)

Прототип:

Int get_Database(Head, int)

Пример вызова:

Valid_file = get_Database(head, 1)

Описание переменных:

Описание переменных приведено в таблице 25.

Таблица 25. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	HEAD	Head*	Указатель на голову
Формальная	MODE	int	Вид чтения с файла (по умолчанию или выбранный пользователем)
Локальная	p	Node*	Очередной элемент списка
Локальная	slen	int	Длина очередной строки
Локальная	i	int	Количество строк в файле
Локальная	flag	int	Проверка на выделение памяти
Локальная	Valid_file	int	Переменная для отслеживания как считался файл
Локальная	sep	char	Сепаратор
Локальная	S1	char	Очередная строка в файле
Локальная	S2	Char **	“Массив” из полей очередной структуры

5.21 Функция write_to_file

Назначение:

Функция для записи в файл

Прототип:

Int write_to_file(Head, int)

Пример вызова:

Valid_file = write_to_file(head, 1)

Описание переменных:

Описание переменных приведено в таблице 26.

Таблица 26. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	HEAD	Head*	Указатель на голову
Формальная	MODE	int	Вид записи в файл (по умолчанию или выбранный пользователем)
Локальная	File	FILE*	Указатель на файл
Локальная	Valid_file	Int	Переменная для отслеживания успешно ли прошла запись в файл
Локальная	Path	Char*	Указатель на строку (название файла)

5.22 Функция edit_node

Назначение:

Функция для редактирования узла

Прототип:

Void edit_node(Head)

Пример вызова:

edit_node (head)

Описание переменных:

Описание переменных приведено в таблице 27.

Таблица 27. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	HEAD	Head*	Указатель на голову
Локальная	Change_int	int	Критерий поиска с целыми числами
Локальная	Change_str	str	Критерий поиска с строками
Локальная	Change_float	float	Критерий поиска с вещественными
Локальная	Variant1	int	Номер элемента списка, который пользователь намеревается редактировать
Локальная	Variant2	int	Выбор пункта меню
Локальная	Exit_flag	int	Флаг на выход из switch
Локальная	Temp_node	Node*	Указатель на очередную структура для редактирования

5.23 Функция search

Назначение:

Вспомогательная функция для поиска search_managers

Прототип:

Head search(Head, manager, int, int *)

Пример вызова:

Search_result = search(HEAD, search_param, 1, bl)

Описание переменных:

Описание переменных приведено в таблице 28.

Таблица 28. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	HEAD	Head*	Указатель на голову
Формальная	Search_param	manager	Новые данные для структуры
Формальная	field	int	Поиск по определенному
Формальная	bl	Int *	Проверка на выделение динамической памяти
Локальная	Search_result	Head *	Указатель на голову нового списка
Локальная	Temp_node	Node *	Указатель на очередную структуру для редактирования
Локальная	Cr_node	Node *	Копия temp_node

5.24 Функция search_managers

Назначение:

Функция для поиска элементов по полям списка

Прототип:

```
void search(Head, int*)
```

Пример вызова:

```
= search_managers(HEAD, search_param, 1, bl)
```

Описание переменных:

Описание переменных приведено в таблице 29.

Таблица 29. Описание переменных функции

Вид переменной	Имя переменной	Тип	Назначение
Формальная	HEAD	Head*	Указатель на голову
Формальная	Search_param	manager	Новые данные для структуры
Формальная	field	int	Поиск по определенному
Формальная	bl	Int *	Проверка на выделение динамической памяти
Локальная	Search_list	Head *	Указатель на голову нового списка
Локальная	temp	Head *	Указатель на голову нового списка (копия)
Локальная	variant	int	Выбор по какому полю искать
Локальная	Search_param	manager	Новые данные для структуры
Локальная	Exit_flag	char	Переменная для выхода из цикла
Локальная	flags	Unsigned char	Массив флагов

6. Текст программы

6.1 main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../include/common.h"
#include "../include/dbl_list.h"
#include "../include/get.h"

int main()
{
    dbl_list();
    return 0;
}
```

6.2 dbl_list.c

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include "../include/dbl_list.h"
#include "../include/get.h"
#include "../include/const.h"
#include <conio.h>

Head *make_head(int *bl) // Создание головы
{
    Head *ph=NULL;
    ph=(Head*)malloc(sizeof(Head));
    if(ph!=NULL)
    {
        ph->count=0;
        ph->first = NULL;
        ph->last = NULL;
    } else *bl = 0;
    return ph;
}

void print_managers(Head *my_head)
{
    Node *p;
    int i;
    printf("|Name of product| price | Country| Availability| Standart|\n");
    printf("_____|_____||_____||_____||_____||\n");
    p = my_head->first;
    for (i = 0; i < my_head->count; i++)
    {
        printf("|%15.15s|%9.2f|%13.13s|%13.13s|%12.12s|%11.11s|\n", (p->info).name_product, (p->info).price, (p->info).country, (p->info).availability ? "Is available" : "Not available", (p->info).verify ? "Verified" : "Not verified", (p->info).pr_month ? "Popular" : "Not popular");
        printf("_____|_____||_____||_____||_____||\n");
        p = p->next;
    }
}

char *get_category() // Выбор категории. Возвращает число (пункт в списка категорий)
{
    char c; // Переменная для "fflush"
    int q; // Выбор пункта из списка категорий
    int i; // Индекс категории
    for (i = 0; i < cat_num; i++)
        printf("%d. %s\n", i+1, categories[i]);
    do
    {
        scanf("%d", &q);
        if (q <= 0 || q > (int)sizeof(categories)/sizeof(char*))
            printf("You entered an invalid value, try again: \n");
    } while (q <= 0 || q > (int)sizeof(categories)/sizeof(char*));
    while ((c = getchar()) != '\n' && c != EOF);
    printf("Your choice %s\n", categories[q-1]);
    return categories[q-1];
}

void fill_node(SHOP *list, int *bl) // Ввод очередной структуры
{
    system("cls");
    int buff;
    list->name_product = (char*)malloc(MAXLEN*sizeof(char));
    list->country = (char*)malloc(MAXLEN*sizeof(char));
```

```

if (list->name_product && list->country)
{
    printf("Enter name of product\n");
    list->name_product = get_string(&b1);
    do
    {
        printf("Enter price\n");
        list->price = get_float();
        if(list->price < 0)
            puts("You entered an invalid number");
    } while (list->price < 0);
    printf("Select the country: \n");
    list->country = get_category();

    do
    {
        printf("Available? 1 - YES, 0 - NO \n");
        buff = get_int();
        if(buff != 1 && buff != 0)
            puts("You entered an invalid number");
        else list->availability = buff;
    } while (buff != 1 && buff != 0);
    do
    {
        printf("Verified? 1 - YES, 0 - NO \n");
        buff = get_int();
        if(buff != 1 && buff != 0)
            puts("You entered an invalid number");
        else list->verify = buff;
    } while (buff != 1 && buff != 0);
    do
    {
        printf("Product of the month? 1 - YES, 0 - NO \n");
        buff = get_int();
        if(buff != 1 && buff != 0)
            puts("You entered an invalid number");
        else list->pr_month = buff;
    } while (buff != 1 && buff != 0);
    } else *b1 = 0;
}

Node *create_node(int *b1) // Создание узла
{
    Node *new_node=NULL;
    new_node = (Node*)malloc(sizeof(Node));
    if(new_node)
    {
        fill_node(&(new_node->info), b1);
    } else *b1 = 0;
    new_node->prev = NULL;
    new_node->next = NULL;
    return new_node;
}

void *add_first(Head *my_head, Node *new_node) // Добавление элемента в начал
o
{
    if(my_head&&new_node)
    {
        if (!(my_head->count))
            my_head->last = new_node;
        else
        {
            (my_head->first)->prev = new_node;
            new_node->next = my_head->first;
        }
        my_head->first = new_node;
        my_head->count++;
    }
}

void *add_last(Head *my_head, Node *new_node) // Добавление элемента в конец

```

```

{
    if(my_head&&new_node)
    {
        if (!(my_head->count))
            my_head->first = new_node;
        else
        {
            (my_head->last)->next = new_node;
            new_node->prev = my_head->last;
        }
        my_head->last = new_node;
        my_head->count++;
    } else;
}

void insert(Head *my_head, Node *new_node) // Добавление элемента в n-
ую позицию
{
    int i, // Переменная в цикле
        pos; // Выбор позиции пользователем
    Node *p;

    do
    {
        printf("what is the position? [1 to %d]\n", my_head->count+1);
        pos = get_int();
    } while (pos < 1 || pos > my_head->count+1);

    if(my_head&&new_node)
    {
        if (!(my_head->count)) // Если в списке ноль элементов
        {
            my_head->first = new_node;
            my_head->last = new_node;
            my_head->count++;
        }
        else if (pos > 0 && pos < my_head->count+2)
        {
            if (pos == 1)
                add_first(my_head, new_node);
            else if (pos == my_head->count + 1)
                add_last(my_head, new_node);
            else {
                p = my_head->first;
                for (i = 1; i < pos - 1; i++) // Добираемся до позиции pos-1
                    p = p->next;
                new_node->prev = p;
                new_node->next = p->next;
                (p->next)->prev = new_node;
                p->next = new_node;
                my_head->count++;
            }
        }
    }
}

void add_item(Head *HEAD, int *b1) // Добавление элемента в список
{
    Node *p=NULL;
    char c;
    do
    {
        printf("1 - Add node to start\n2 - Add node to end\n3 - Insert node\n\n");
        Press Enter to stop\n");
        c = getch();
        if (c != 13) p = create_node(b1);
        switch (c)
        {
            case 49:
                add_first(HEAD, p);
                break;
            case 50:

```

```

        add_last(HEAD, p);
        break;
    case 51:
        insert(HEAD, p);
        break;
    case 13:
        break;
    default:
        puts("Error, try again.\n");
    }
} while (c != 13);
}

// Копирование узла
Node *copy_node(Node *NODE, int *b1)
{
    Node *p=NULL;

    p = (Node*)malloc(sizeof(Node));
    (p->info).name_product = (char*)malloc(MAXLEN*sizeof(char));
    (p->info).country = (char*)malloc(MAXLEN*sizeof(char));

    if((p->info).name_product!=NULL && (p->info).country!=NULL)
    {
        strcpy((p->info).name_product, (NODE->info).name_product);
        strcpy((p->info).country, (NODE->info).country);
        (p->info).availability = (NODE->info).availability;
        (p->info).price = (NODE->info).price;
        (p->info).verify = (NODE->info).verify;
        (p->info).pr_month = (NODE->info).pr_month;
    } else *b1 = 0;

    return p;
}

void swap(Head *HEAD, int first, int second)
{
    Node *p_one, *p_two;
    Node *buff_one, *buff_two;
    int i;

    p_one = HEAD->first;
    for (i = 1; i < first - 1; i++)
        p_one = p_one->next; // Доходим до нужной позиции
    p_two = HEAD->first;
    for (i = 1; i < second - 1; i++) //Аналогично
        p_two = p_two->next;

    if (first != 1)
    {
        // Делаем саму перестановку
        buff_one = p_one->next;
        buff_two = p_two->next;
        p_one->next = buff_two;
        p_two->next = buff_one;
        buff_two->prev = p_one;
        buff_one->prev = p_two;
        p_one = buff_one->next;
        p_two = buff_two->next;
        buff_one->next = p_two;
        buff_two->next = p_one;
        if (buff_two == HEAD->last)
        {
            HEAD->last = buff_one;
        } else p_two->prev = buff_one;
    }
    else
    {
        buff_two = p_two->next;
        buff_one = p_one;
        HEAD->first = buff_two;
        p_two->next = buff_one;
    }
}

```



```

        buff_two->prev = p_one;
        buff_one->prev = p_two;
        p_one = buff_one->next;
        p_two = buff_two->next;
        buff_one->next = p_two;
        buff_two->next = p_one;
        if (buff_two == HEAD->last)
        {
            HEAD->last = buff_one;
        }
        else p_two->prev = buff_one;
    }
}

// Высвобождение памяти узла
Node *clean_node(Node *node)
{
    free((node->info).name_product);
    (node->info).name_product = NULL;
    free(node);
    return NULL;
}

// Удаление узла
void remove_node(Head *my_head)
{
    Node *p;    // Указатели требуемые для удаления узла
    int i;      // Переменная в цикле
    pos;        // Выбор позиции
    char c;

    printf("Want to see a list of notes? Enter one if yes or another key to c
ontinue\n");
    c = getch();
    if (c == 49)
        print_managers(my_head);
    do
    {
        do
        {
            printf("Delete item numbered [From 1 to %d]: ", my_head->count);
            pos = get_int();
        } while (pos < 1 || pos > my_head->count);
        p = my_head->first;
        if (my_head->count > 1)
        {
            for (i = 1; i < pos - 1; i++)
                p = p->next;
            if (pos == 1)
            {
                my_head->first = p->next;
                (p->next)->prev = NULL;
            }
            else if (pos == my_head->count)
            {
                my_head->last = p->prev;
                (p->prev)->next = NULL;
            }
            else
            {
                (p->prev)->next = p->next;
                (p->next)->prev = p->prev;
            }
        }
        else
        {
            my_head->first = NULL;
            my_head->last = NULL;
        }
        my_head->count--;
        clean_node(p);
    }
}

```

```

        if (my_head->count > 0)
        {
            printf("Удалить еще? Введите 1 - Да, любую другую клавишу - Нет\n");
            c = getch();
        }
        else
            c = 49;
    } while (c == 49 && my_head->count > 0);
}

// Фильтр. Возвращает голову нового списка
Head *selected(Head *my_head, int *bl)
{
    Head *NEW_HEAD = NULL;
    Node *p = NULL;
    NEW_HEAD = make_head(bl);

    p = my_head->first;
    for (int i=0; i<my_head->count; i++)
    {
        if (((p->info).availability == 1) && ((p->info).verify == 0))
            add_last(NEW_HEAD, copy_node(p, bl));
        p = p->next;
    }
    puts("Success!");
    return NEW_HEAD;
}

// Высвобождение памяти списка
Head *clean_list(Head *HEAD)
{
    Node *p, *temp;
    int i;

    p = HEAD->first;
    HEAD->count = 0;
    for (i = 0; i < HEAD->count; i++)
    {
        temp = p;
        p = p->next;
        temp->next = NULL;
        temp->prev = NULL;
        temp = clean_node(temp);
    }
    free(HEAD);
    return NULL;
}

```

6.3 get.c

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include "../include/get.h"

char *get_string() // Возвращает указатель на введенную строку
{
    char c;
    char *string = (char*)malloc(MAXLEN*sizeof(char)); // очередной символ
    int i = 0; // Указатель на строку
    // Индекс очередного
    символа
    if (string != NULL)
    {
        do
        {
            while ((c = getchar()) != '\n' && i < MAXLEN-1) string[i++] = c;
            string[i] = '\0';
            if (i < 1) printf("Error. You entered empty string. Please, try
again.\n");
        } while (i < 1);
        if (i >= MAXLEN - 1)
            while ((c = getchar()) != '\n' && c != EOF);
    }
    return string;
}

int get_int() // Ввод числа
{
    char line[MAXLEN]; // Буффер
    char curChar = ' '; // Последний символ
    int temp, result;
    do {
        line[MAXLEN - 1] = '\n';
        fgets(line, MAXLEN - 1, stdin);
        temp = sscanf(line, "%d%c", &result, &curChar);
        temp = !temp || temp < 0 || (curChar != '\n' && curChar != ' ');
        if (temp)
            printf("Error reading number. Please, try again.\n");
    } while (temp); // Не число на самом деле
    if (line[MAXLEN - 1] != '\n') //clear all input
        while ((curChar = getchar()) != '\n' && curChar != EOF);
    return result;
}

float get_float() // Ввод вещественного числа
{
    char line[MAXLEN]; // Буффер
    char curChar = ' ';
    int temp;
    float result;
    do {
        line[MAXLEN - 1] = '\n';
        fgets(line, MAXLEN - 1, stdin);
        temp = sscanf(line, "%f%c", &result, &curChar);
        temp = !temp || temp < 0 || (curChar != '\n' && curChar != ' ');
        if (temp)
            printf("Error reading number. Please, try again.\n");
    } while (temp); // Не число на самом деле
    if (line[MAXLEN - 1] != '\n') // избавление от мусора
        while ((curChar = getchar()) != '\n' && curChar != EOF);
    return result;
}
```

6.4 w_file.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>
#include "../include/dbl_cycle_list.h"
#include "../include/dbl_list.h"
#include "../include/get.h"

void clear_str_array(char **str, int n)
{
    int i;
    for(i = 0; i < n; i++)
    {
        free(str[i]);
        str[i] = NULL;
    }
    free(str);
    str = NULL;
}

// Конвертация в узел
Node *convert_to_node(char **s2)
{
    Node *p = NULL;
    int len1, len2; // Длина строки категории и описания соответственно
    p = (Node*)malloc(sizeof(Node));
    p->next = NULL;
    p->prev = NULL;
    char str1[] = "Income";
    len1 = strlen(s2[1]);
    len2 = strlen(s2[2]);

    (p->info).category=(char*)malloc((len1+1)*sizeof(char));
    (p->info).description=(char*)malloc((len2+1)*sizeof(char));

    if(((p->info).category!=NULL)&&((p->info).description!=NULL))
    {
        if(strcmp(str1, s2[0]))
            (p->info).expenses_income = 2;
        else
            (p->info).expenses_income = 1;
        strcpy((p->info).category, s2[1]);
        strcpy((p->info).description, s2[2]);
        (p->info).money = atoi(s2[3]);
    }
    else
    {
        puts("Out of memory! Program terminated");
        p = NULL;
    }
    return p;
}

// Разделение строки на подстроки
char **simple_split(char *str, int length, char sep)
{
    char **str_array = NULL;
    int i, j,
        k,
        m; // Количество сепараторов в строке
    int key, // Проверка на выделение памяти
        count; // Количество строк, которым была выделена память
    for(j = 0, m = 0; j < length; j++)
        if(str[j] == sep) m++;

    key = 0;
    str_array = (char**)malloc((m+1)*sizeof(char*));
    if(str_array != NULL)
    {
```

```

    for(i = 0, count = 0; i <= m; i++, count++)
    {
        str_array[i] = (char*)malloc(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++;
            }
        }
    } else clear_str_array(str_array, count);
}
return str_array;
}

// чтение со строки
int get_database(Head *HEAD, int MODE)
{
    Node *p;
    int slen,          // Длина очередной строки
        i,             // Количество строк в файле
        flag = 1,      // Проверка на выделение памяти
        valid_file;    // -1 - чтение не вышло. 1 - чтение прошло успешно. 2 -
// Файл пустой. 0 - Не удалось открыть файл
    char sep;          // Сепаратор
    char s1[MAXSTR];   // Очередная строка в файле
    char **s2 = NULL;  // Массив из полей очередной структуры

    FILE *df;
    if (MODE) // чтение из нового файла
    {
        char *path;
        puts("Type path to file or his name: ");
        path = get_string();
        df = fopen(path, "r");
    }
    else df = fopen("database.txt", "r"); // чтение из файла по умолчанию
    if(df != NULL)
    {
        sep=';';      // Сепаратор
        i = 0;
        while(fgets(s1, MAXSTR, df) != NULL && flag)
        {
            slen = strlen(s1);
            if(s1[slen-1] == '\n')
                s1[slen-1] = '\0';
            else
                s1[slen] = '\0';
            slen = strlen(s1);
            s2 = simple_split(s1, slen, sep);
            if(s2 != NULL)
            {
                p = convert_to_node(s2);
                add_last(HEAD, p);
                i++;
            }
            clear_str_array(s2, 4);
        }
    }
}

```

```

        else
        {
            flag = 0;
            valid_file = -1;
            puts("Row data not available!");
        }
    }
    if (fclose(df) != EOF)
    {
        if (i == 0)
            valid_file = 2;
        else
            valid_file = 1;
    }
    else;
}
else
    valid_file = 0;
return valid_file;
}

// чтение со строки
int cycle_get_database(Head *HEAD, int MODE)
{
    Node *p;
    int slen,          // Длина очередной строки
        i,             // Количество строк в файле
        flag = 1,      // Проверка на выделение памяти
        valid_file;    // -1 - чтение не вышло. 1 - Чтение прошло успешно. 2 -
// файл пустой. 0 - Не удалось открыть файл
    char sep;          // Сепаратор
    char s1[MAXSTR];   // Очередная строка в файле
    char **s2 = NULL;  // Массив из полей очередной структуры

    FILE *df;
    if (MODE) // чтение из нового файла
    {
        char *path;
        puts("Type path to file or his name: ");
        path = get_string();
        df = fopen(path, "r");
    }
    else df = fopen("database.txt", "r"); // чтение из файла по умолчанию
    if (df != NULL)
    {
        sep = ';'; // Сепаратор
        i = 0;
        while (fgets(s1, MAXSTR, df) != NULL &&

```

6.5 common.c

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <conio.h>
#include "../include/dbl_list.h"
#include "../include/common.h"
#include "../include/dbl_cycle_list.h"
#include "../include/get.h"
#include "../include/w_file.h"

// Меню
int Menu(int q)
{
    int Q;
    system("cls");
    puts("MENU");
    switch(q)
    {
        case 0:
            puts("1 - Input data");
            puts("2 - Output data");
            puts("3 - Actions with the database");
            puts("4 - Filter");
            puts("5 - Output result");
            puts("6 - Help");
            puts("0 - Exit");
            break;
        case 1:
            puts("1 - Input from console");
            puts("2 - Input from file");
            puts("0 - Come back");
            break;
        case 12:
            puts("1 - Input from default file (database.txt)");
            puts("2 - Select a file");
            puts("0 - Come back");
            break;
        case 3:
            puts("1 - Swap any items");
            puts("2 - Remove any items");
            puts("3 - Sort database");
            puts("4 - Add item");
            puts("5 - Save changes to file");
            puts("0 - Come back");
            break;
        case 5:
            puts("1 - Console output");
            puts("2 - Output in file");
            puts("0 - Come back");
            break;
        case 52:
            puts("1 - Write to file by default(output.txt)");
            puts("2 - Select a file");
            puts("0 - Come back");
    }

    printf("Select menu item - ");
    scanf("%d", &Q);
    printf("\n");
    fflush(stdin);
    return Q;
}

// справка
void Help()
{
    system("cls");
    printf("\tHelp\n");
}
```

```

    printf(" First you need to enter notes (your income or expenses). To enter the input press 1\n");
    printf(" Notes have the following characteristics:\n");
    printf(" Income or Cost. Category. Description of the note. Amount of money spent\n");
    printf(" For actions with the list, select item 3. Filter the list by the specified categories - item 4. Display the result - 5.\n");
}

int dbl_list()
{
    Head *HEAD = NULL, // Голова списка
    *NEW_HEAD = NULL; // Голова результирующего списка
    Node *p = NULL; // Очередной узел
    int Q, // Выбор пункта меню
    Q3, Q1, Q12, Q5, Q52, // Выбор пункта подменю
    output = 0, // Проверка на ввод входных данных
    b1 = 1, // Проверка на выделение памяти
    valid_file, // Валидация файла
    first, // Элемент сортировки
    buff, // Для swap first и second
    second; // Элемент сортировки
    char c = 0; // Ввод числа или Enter
    HEAD = make_head(&b1); // Голова

    if(b1)
    {
        do
        {
            Q = Menu(0);
            switch (Q)
            {
                case 1: //input
                    output = 0;
                    if (HEAD->count)
                    {
                        HEAD = clean_list(HEAD);
                        HEAD = make_head(&b1);
                        if (NEW_HEAD)
                            NEW_HEAD = clean_list(NEW_HEAD);
                    }
                do
                {
                    Q1 = Menu(1);
                    switch (Q1)
                    {
                        case 1:
                            add_item(HEAD, &b1);
                            printf("Successful input.\n");
                            Q1=0;
                            break;
                        case 2:
                            do
                            {
                                Q12 = Menu(12);
                                switch (Q12)
                                {
                                    case 1:
                                        valid_file = get_database(HEAD, 0);
                                        if (valid_file == 1)
                                            printf("Successful input.\n");
                                        else if (valid_file == 0)
                                            printf("Error: Nonexistent file.\n");
                                        else if (valid_file == 2)
                                            printf("Error, file is empty.\n");
                                        else
                                            printf("Error reading from file.\n");
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```



```

        Q12 = 0;
        break;
    case 2:
        valid_file = get_database(HEAD, 1);
        if (valid_file==1)
            printf("Successful input.\n");
        else if (valid_file == 0)
            printf("Error: Nonexistent fi
        else if (valid_file == 2)
            printf("Error, file is empty.
        else
            printf("Error reading from fi

        Q12 = 0;
        break;
    case 0:
        break;
    default:
        puts("Error, try again.\n");
    }
} while (Q12 != 0 && b1);

    Q1 = 0;
    break;
    case 0:
        break;
    default:
        puts("Try again.");
    }
} while(Q1 != 0 && b1);

    break;
    case 2: //output
        if (HEAD->count)
            print_managers(HEAD);
        else
            printf("No input to print!\n");
        break;
    case 3:
        if (HEAD->count != 0)
        {
            do
            {
                Q3 = Menu(3);
                switch (Q3)
                {
                    case 1:
                        if (HEAD->count > 1)
                        do
                        {
                            do
                            {
                                system("cls");
                                printf("Do you want see list
                                otherwise\n");
                                c = getch();
                                if (c == 49)
                                    print_managers(HEAD);
                                printf("Enter first item numb
                                first = get_int();
                                printf("Enter second item num
                                second = get_int();
                            } while ((first<1 || second>HEAD->
                                >count) || (second<1 || first>HEAD->count));
                                if (first>second)
                                    {

```

```

buff = first;
first = second;
second = buff;
    }
    swap(HEAD, first, second);
    puts("Once more swap? Press Enter

- No, press any key - Yes");
    c = getch();
    } while (c != 13);
else
{
    Q3=0;
    puts("The list must have more than on
e item.");
    }
    break;
case 2:
    remove_node(HEAD);
    break;
case 3:
    sort(HEAD);
    puts("Successfully sorted.");
    system("pause");
    break;
case 4:
    add_item(HEAD, &b1);
    break;
case 5:
    valid_file = write_to_file(HEAD, 1);
    if (valid_file == 1)
        puts("Successful write to file");
    else
        puts("Error write to file");
    system("pause");
    break;
case 0:
    Q3=0;
    break;
default:
    puts("Try again!");
    break;
    }
    } while (Q3 != 0 && HEAD->count && b1);
}
else
    printf("No input to actions!\n");
break;
case 4: //process
    if (HEAD->count)
    {
        NEW_HEAD = selected(HEAD, &b1);
        output = 1;
    }
    else
        printf("No input to process!\n");
break;
case 5: //output result
    if (output)
    do
    {
        Q5 = Menu(5);
        switch (Q5)
        {
            case 1:
                if (NEW_HEAD-
>count) print_managers(NEW_HEAD);
                else printf("Managers not found.\n");
                Q5=0;
                break;
            case 2:
                do
                {

```

```

        Q52 = Menu(52);
        switch (Q52)
        {
            case 1:
                if (NEW_HEAD->count) valid_file = write_to_file(NEW_HEAD, 0);
                else printf("Managers not found.\n");
                if (valid_file == 1)
                    puts("Successful write to file.");
                else
                    puts("Error writing to file.");
                Q52 = 0;
                break;
            case 2:
                if (NEW_HEAD->count) valid_file = write_to_file(NEW_HEAD, 1);
                else printf("Managers not found.\n");
                if (valid_file == 1)
                    puts("Successful write to file.");
                else
                    puts("Error writing to file.");
                Q52 = 0;
                break;
            case 0:
                break;
            default:
                puts("Error, try again.\n");
        }
    } while (Q52 != 0 && b1);

    Q5=0;
    break;
case 0:
    break;
default:
    puts("Try again.");
}
} while(Q5 != 0 && b1);
else
    printf("No processed data to output!\n");
    break;
case 6: //help
    Help();
    break;
case 0:
    break;
default:
    printf("Error! Try again!\n");
    break;
}
    system("pause");
} while (Q && b1);
} else printf("Error memory");
// Высвобождение памяти
if (HEAD->count != 0)
    HEAD = clean_list(HEAD);
if (NEW_HEAD)
    NEW_HEAD = clean_list(NEW_HEAD);
return 0;
}

```

6.6 common.h

```
#ifndef COMMON_H
#define COMMON_H

int Menu(int q); // Меню

void Help(); // Справка

int dbl_list();

int dbl_cycle_list();

#endif

6.8 const.h
#ifndef CONST_H
#define CONST_H
#define cat_num 5

//Категории затрат/доходов
const char *categories[cat_num] =
{
    "Food",
    "transport",
    "Clothes",
    "Social",
    "other"
};

#endif
```

6.7 dbl_list.h

```
#ifndef LIST_H
#define LIST_H
#define MAXSTR 128

typedef struct
{
    // Описание полей
    int expenses_income;
    char *category;    // Категория затрата/дохода
    char *description; // Описание (комментарий) к покупке
    float money;       // Количество потраченных/заработанных денег
} manager;

typedef struct manager_elem
{
    manager info;
    struct manager_elem *next;
    struct manager_elem *prev;
} Node; // Очередной элемент (узел) списка

typedef struct
{
    int count;
    Node *first;
    Node *last;
} Head; // Голова списка

#endif // LIST_H

Head *make_head(int *bl); // Создание головы

Node *create_node(int *bl); // Создание узла

char *get_category(); // Ввод категории

void fill_node(manager *list, int *bl);

void add_item(Head *HEAD, int *bl); // Добавление элемента в список
```

```

void print_managers(Head *my_head);           // Вывод заметок

void *add_first(Head *my_head, Node *new_node); // добавление элемента в нача
ло

void *add_last(Head *my_head, Node *new_node); // добавление элемента в конец

void insert(Head *my_head, Node *new_node); // добавление элемента в n-
ую позицию

Node *copy_node(Node *NODE, int *bl);         // копирование элемента списк
а и возвращает копию

void swap(Head *HEAD, int first, int second);

void remove_node(Head *my_head);

int compare(Node *left, Node *right, int type); // Сравнение элементо
в списка для сортировки

void sort(Head *HEAD);                         // Сортировка по цене
или доходам/затратам

Head *selected(Head *my_head, int *bl);        // Фильтр по заметкам
по минимальной цене и категории

Node *clean_node(Node *node);

Head *clean_list(Head *HEAD);

```

6.8 get.h

```
#ifndef GET_H
#define GET_H
#define MAXLEN 25

char *get_string();           // Ввод строки
int  get_int();              // Ввод целочисленного числа
float get_float();           // Ввод вещественного числа

#endif // GET_H
```

6.9 w_file.h

```
#ifndef W_FILE_H
#define W_FILE_H
#define cat_num 5

char **simple_split(char *str, int length, char sep); // Разбиение строк на
д подстроки по сепаратору
int get_database(Head *HEAD, int MODE);              // Чтение из файла
int cycle_get_database(Head *HEAD, int MODE);        // Чтение из фа
йла
int write_to_file(Head *HEAD, int MODE);             // Запись в файл
Node *convert_to_node(char **s2);                   // Конвертация масси
ва строк в элемент списка

#endif
```

7. Пример работы программы

При выполнении программы получены результаты, совпадающие со значениями, приведенными в Таблице 1. Ошибок не обнаружено. Пример протокола выполнения программы приведен на Рис.3.

Рисунок 3. Пример работы программы

```
Главное меню
1 - Ввод
2 - Контрольный вывод ...
3 - Действия со списком
4 - Фильтр
5 - Вывод результата
6 - Справка
0 - Выход
Введите номер пункта - Выберите пункт меню - 2

|Затраты/Доходы|Категория|Описание|Кол-во денег|
|-----|-----|-----|-----|
|Затраты|Транспорт|Metro|55|
|Затраты|Транспорт|Bus|55|
|Затраты|Питание|HotDog|75|
Для продолжения нажмите любую клавишу . . .
```


Заключение

При выполнении курсовой работы получены практические навыки поэтапного решения содержательной задачи, связанной с использованием структур, двусвязных линейных списков на языке программирования «C/C++».