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 | Дементьев Д.П. |
|  |  |
| Проверил к.т.н., доцент | Сискович Т.И. |

Санкт-Петербург, 2020

Оглавление

[Цель 3](#_Toc40625444)

[1. Задание 3](#_Toc40625445)

[2. Уточнение задания 3](#_Toc40625446)

[3. Контрольные примеры](#_Toc40625447) **[Ошибка! Закладка не определена.](#_Toc40625447)**

[4. Краткое описание алгоритма 10](#_Toc40625448)

[5. Функции 11](#_Toc40625449)

[5.1 Главная функция 11](#_Toc40625450)

[5.2 Функция dbl\_list 12](#_Toc40625451)

[5.3 Функция add\_item 13](#_Toc40625452)

[5.4 Функция get\_string 15](#_Toc40625453)

[5.5 Функция get\_int 16](#_Toc40625454)

[5.6 Функция get\_float 17](#_Toc40625455)

[5.7 Функция get\_category 18](#_Toc40625456)

[5.8 Функция fill\_node 19](#_Toc40625457)

[5.9 Функция make\_head 20](#_Toc40625458)

[5.10 Функция print\_managers 21](#_Toc40625459)

[5.11 Функция selected 22](#_Toc40625460)

[5.12 Функция add\_first 23](#_Toc40625461)

[5.13 Функция add\_last 24](#_Toc40625462)

[5.14 Функция insert 25](#_Toc40625463)

[5.15 Функция swap 26](#_Toc40625464)

[5.16 Функция remove\_node 27](#_Toc40625465)

[5.17 Функция copy\_node 28](#_Toc40625466)

[5.18 Функция create\_node 29](#_Toc40625467)

[5.19 Функция Menu 30](#_Toc40625468)

[5.20 Функция get\_database 31](#_Toc40625469)

[5.21 Функция write\_to\_file 32](#_Toc40625470)

[6. Текст программы 36](#_Toc40625471)

[6.1 main.c 36](#_Toc40625472)

[6.2 dbl\_list.c 37](#_Toc40625473)

[6.3 get.c 43](#_Toc40625474)

[6.4 w\_file.c 44](#_Toc40625475)

[6.5 common.c 47](#_Toc40625476)

[6.6 common.h 52](#_Toc40625477)

[6.7 dbl\_list.h 53](#_Toc40625478)

[6.8 get.h 55](#_Toc40625479)

[6.9 w\_file.h 55](#_Toc40625480)

[7. Пример работы программы 56](#_Toc40625481)

[Заключение 57](#_Toc40625482)

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

При выполнении задания необходимо учитывать:

1. Вводимый текст не может быть пустым.
2. Максимальная длина вводимой строки – 24 символа.
3. Пользователь может выбрать работу со стеком или очередью перед выполнением программы.
4. При смене режима работы список не очищается, а только меняется режим добавления/удаления элементов (например, добавление в начало списка для стека и в конец списка - для очереди).
5. Предметная область - база репетиторов, в которой есть такие критерии как имя, цена за час, предмет преподавания, квалификация, рейтинг.

Для данной предметной области были выбраны следующие поля структур:

Для структуры tutor:

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип** | **Назначение** |
| name | char\* | Имя преподавателя |
| subject | char\* | Предмет преподавания |
| qual | char | Квалификация преподавателя |
| price | int | Цена за час занятия |
| rating | float | Рейтинг преподавателя |

Для структуры Node:

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип** | **Назначение** |
| next | tutor\_elem\* | Указатель на следующую структуру |
| info | tutor | Вложенная структура с информацией об объекте |
| prev | tutor\_elem\* | Указатель на предыдущую структуру |

Для структуры Head:

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип** | **Назначение** |
| count | int | Количество узлов в списке |
| first | Node\* | Указатель на адрес первого элемента списка |
| last | Node\* | Указатель на адрес последнего элемент в списке |

Поле subject определяется введенным пользователем числом от 1 до 5, обозначающим номер предмета(по желанию можно редактировать этот список):

1. Maths

2. Computer science

3. English

4. Russian

5. Physics

Меню программы должно иметь следующую иерархию:

1 - Ввод исходных данных

2.1 - Ввод с консоли

2.2 - Ввод из файла

2.2.1 - Ввод из файла по умолчанию (database.txt)

2.2.2 - Выбрать произвольный файл

2.2.0 - В главное меню

2.0 - Назад

2 - Вывод исходных данные

3 - Действия с базой данных

3.1 - Поменять элементы местами

3.2 - Удаление элементов

3.3 - Сортировка списка

3.4 - Добавить элемент к списку

3.5 - Сохранить изменённый список на диске

3.0 - Назад

4 - Сортировка исходных данных

5 - Вывод отсортированных данных

5.1 - Вывод в консоль

5.2 - Вывод в файл

5.2.1 - Вывод в файл по умолчанию (output.txt)

5.2.2 - Вывод в произвольный файл

5.2.0 - В главное меню

5.0 - Назад

6 - Справка

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 пункта пользователю выводится краткая справка о работе программы

При выборе 0 пункта происходит выход программы.

# 3. Контрольные примеры

Контрольные примеры приведены в таблице 1 и 2 соответственно.

Таблица 1. Контрольные примеры

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Поля структуры | | | | | Критерии | |
| **Предмет:** математика**Мин. рейтинг:** 3.0**Макс. цена:** 1000 | **Предмет:** физика**Мин. рейтинг:** 4.0**Макс. цена:** 400 |
| Имя | Предмет | Цена | Рейтинг | Квал. | Вывод | |
| 1 | Alex | Maths | 900 | 5 | A | Исх. структура | - |
| 2 | Vlad | Maths | 350 | 3 | C | Исх. структура | - |
| 3 | Liza | Physics | 500 | 4 | A | - | - |
| 4 |  | English | 780 | 4 | C | Error. You entered empty string. Please, try again. | |
| 5 | Oleg | Russian | abcd | 3 | B | Error reading number. Please, try again. | |

Таблица 2. Контрольные примеры

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Поля структуры | | | | | Действия с базой данных | |
| -Удалить структуру 2;  -Поменять местами структуры 1 и 2; | -Поменять местами структуры 2 и 3; -Поменять местами структуры 1 и 2; |
| Имя | Предмет | Цена | Рейтинг | Квал. | Порядок вывода структур | |
| 1 | Alex | Maths | 900 | 5 | A | 3, 1 | 3, 1, 2 |
| 2 | Vlad | Maths | 350 | 3 | C |
| 3 | Liza | Physics | 500 | 4 | A |

# 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:**

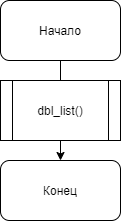
Описание переменных приведено в таблице 6.

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

## D:\Labs\Programming\2semestr\lab4\schems\dbl_list.png5.3 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 сриска |
| Локальная | slen | int | Длина очередной строки |
| Локальная | i | int | Количество строк в файле |
| Локальая | 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 | Valud\_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 | Cp\_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|    Standard|    Popular|\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 категории)

{

    char c;     // Переменная для "fflush"

    int q,      // Выбор пункта из списка категорий

        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(&bl);

        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 \*bl = 0;

}

Node \*create\_node(int \*bl) // Создание узла

{

    Node \*new\_node=NULL;

    new\_node = (Node\*)malloc(sizeof(Node));

    if(new\_node)

    {

        fill\_node(&(new\_node->info), bl);

    } else \*bl = 0;

    new\_node->prev = NULL;

    new\_node->next = NULL;

    return new\_node;

}

void \*add\_first(Head \*my\_head, Node \*new\_node) // Добавление элемента в начало

{

    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 \*bl) // Добавление элемента в список

{

    Node \*p=NULL;

    char c;

    do

    {

        printf("1 - Add node to start\n2 - Add node to end\n3 - Insert node\nPress Enter to stop\n");

        c = getch();

        if (c != 13) p = create\_node(bl);

        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 \*bl)

{

    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 \*bl = 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 continue\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; // Очередной символ

char \*string = (char\*)malloc(MAXLEN\*sizeof(char)); // Указатель на строку

int i = 0; // 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 iput

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 муссора

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 if 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 сепораторов 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; // Сепоратор

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=';'; // Сепоратор

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; // Сепоратор

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=';'; // Сепоратор

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

        bl = 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(&bl);      // Голова

    if(bl)

    {

        do

        {

            Q = Menu(0);

            switch (Q)

            {

                case 1:     //input

                    output = 0;

                    if (HEAD->count)

                    {

                        HEAD = clean\_list(HEAD);

                        HEAD = make\_head(&bl);

                        if (NEW\_HEAD)

                            NEW\_HEAD = clean\_list(NEW\_HEAD);

                    }

                    do

                    {

                        Q1 = Menu(1);

                        switch (Q1)

                        {

                            case 1:

                                add\_item(HEAD, &bl);

                                printf("Successful input.\n");

                                Q1=0;

                                break;

                            case 2:

                                do

                                {

                                    Q12 = Menu(12);

                                    switch (Q12)

                                    {

                                        case 1:

                                            valid\_file = get\_database(HEAD, 0); // database.txt

                                            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); // enter the path of file

                                            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 0:

                                            break;

                                        default:

                                            puts("Error, try again.\n");

                                    }

                                } while (Q12 != 0 && bl);

                                Q1 = 0;

                                break;

                            case 0:

                                break;

                            default:

                                puts("Try again.");

                        }

                    } while(Q1 != 0 && bl);

                    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 of tutors? Press 1 if you want or press any key otherwise\n");

                                                c = getch();

                                                if (c == 49)

                                                    print\_managers(HEAD);

                                                printf("Enter first item number [from 1 to %d]: ", HEAD->count);

                                                first = get\_int();

                                                printf("Enter second item number [from 1 to %d]: ", HEAD->count);

                                                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 one item.");

                                    }

                                    break;

                                case 2:

                                    remove\_node(HEAD);

                                    break;

                                case 3:

                                    sort(HEAD);

                                    puts("Successfully sorted.");

                                    system("pause");

                                    break;

                                case 4:

                                    add\_item(HEAD, &bl);

                                    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 && bl);

                    }

                    else

                        printf("No input to actions!\n");

                    break;

                case 4:     //process

                    if (HEAD->count)

                    {

                        NEW\_HEAD = selected(HEAD, &bl);

                        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 && bl);

                                    Q5=0;

                                    break;

                                case 0:

                                    break;

                                default:

                                    puts("Try again.");

                            }

                        } while(Q5 != 0 && bl);

                    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 && bl);

    } 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 списки

#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 начало

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 списка 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 элементов 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 над подстроки по сепартору

int get\_database(Head \*HEAD, int MODE);                 // Чтение из файла

int cycle\_get\_database(Head \*HEAD, int MODE);                 // Чтение из файла

int write\_to\_file(Head \*HEAD, int MODE);                // Запись в файл

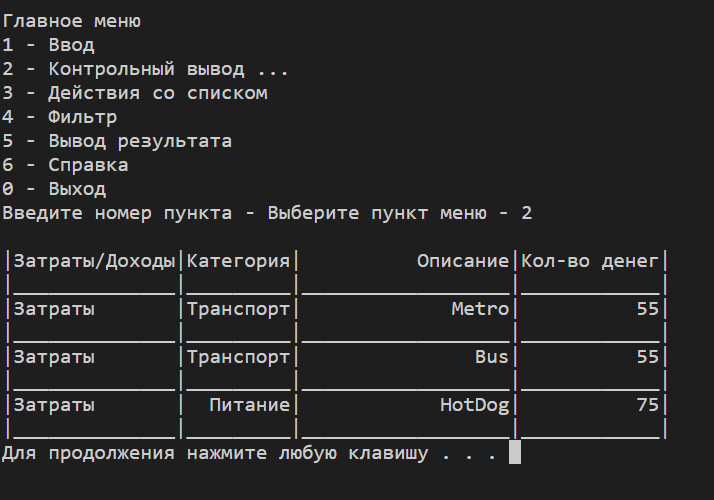
Node \*convert\_to\_node(char \*\*s2);                       // Конвератация массива строк в элемент списка

#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



# Заключение

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