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](#_Toc41668876)

[1. Задание 3](#_Toc41668877)

[2. Уточнение задания 3](#_Toc41668878)

[3. Контрольные примеры 9](#_Toc41668879)

[4. Краткое описание алгоритма 10](#_Toc41668880)

[5. Функции 11](#_Toc41668881)

[5.1 Главная функция 11](#_Toc41668882)

[5.2 Функция dbl\_list 12](#_Toc41668883)

[5.3 Функция add\_item 13](#_Toc41668884)

[5.4 Функция get\_string 15](#_Toc41668885)

[5.5 Функция get\_int 16](#_Toc41668886)

[5.6 Функция get\_float 17](#_Toc41668887)

[5.7 Функция get\_category 18](#_Toc41668888)

[5.8 Функция fill\_node 19](#_Toc41668889)

[5.9 Функция make\_head 20](#_Toc41668890)

[5.10 Функция print\_managers 21](#_Toc41668891)

[5.11 Функция selected 22](#_Toc41668892)

[5.12 Функция add\_first 23](#_Toc41668893)

[5.13 Функция add\_last 24](#_Toc41668894)

[5.14 Функция insert 25](#_Toc41668895)

[5.15 Функция swap 26](#_Toc41668896)

[5.16 Функция remove\_node 27](#_Toc41668897)

[5.17 Функция copy\_node 28](#_Toc41668898)

[5.18 Функция create\_node 29](#_Toc41668899)

[5.19 Функция Menu 30](#_Toc41668900)

[5.20 Функция get\_database 31](#_Toc41668901)

[5.21 Функция write\_to\_file 32](#_Toc41668902)

[5.22 Функция edit\_node 33](#_Toc41668903)

[5.23 Функция search 34](#_Toc41668904)

[5.24 Функция search\_managers 35](#_Toc41668905)

[6. Текст программы 36](#_Toc41668906)

[6.1 main.c 36](#_Toc41668907)

[6.2 dbl\_list.c 37](#_Toc41668908)

[6.3 get.c 56](#_Toc41668909)

[6.4 w\_file.c 57](#_Toc41668910)

[6.5 common.c 60](#_Toc41668911)

[6.6 common.h 70](#_Toc41668912)

[6.7 dbl\_list.h 71](#_Toc41668913)

[6.8 get.h 73](#_Toc41668914)

[6.9 w\_file.h 73](#_Toc41668915)

[7. Пример работы программы 74](#_Toc41668916)

[Заключение 75](#_Toc41668917)

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 фукнции 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. Взять с файла
      1. Взять с файла по умолчанию?
      2. Выбрать файл
      3. Вернутся назад
   2. Ввести с консоли
      1. Добавление в начало

1.2.1.1Выбор дохода или затраты

1.2.1.2Ввод описания

1.2.1.3Ввод количества денег

1.2.1.4Продолжить ввод или нет

* + 1. Добавление в конец

1.2.2.1Выбор дохода или затраты

1.2.2.2Ввод описания

1.2.2.3Ввод количества денег

1.2.2.4Продолжить ввод или нет

* + 1. Добавление в n-ую позицию

1.2.3.1Выбор дохода или затраты

1.2.3.2Ввод описания

1.2.3.3Ввод количества денег

1.2.3.4Вставка в n-ую позицию

1.2.3.5Продолжить 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?

* 1. Добавление элемента

3.4.1 Добавление в начало

3.4.1.1Выбор дохода или затраты

3.4.1.2Ввод описания

3.4.1.3Ввод количества денег

3.4.1.4Продолжить ввод или нет

3.4.2 Добавление в конец

3.4.2.1Выбор дохода или затраты

3.4.2.2Ввод описания

3.4.2.3Ввод количества денег

3.4.2.4Продолжить ввод или нет

* + 1. Добавление в n-ую позицию

3.4.3.1Выбор дохода или затраты

3.4.3.2Ввод описания

3.4.3.3Ввод количества денег

3.4.3.4Вставка в n-ую позицию

3.4.3.5Продолжить ввод или нет

* 1. 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 пункта 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 рождение | 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:**

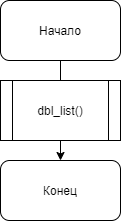
Описание переменных приведено в таблице 6.

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

## D:\Labs\Programming\2semestr\course_work\schems\Untitled Diagram.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 "./include/common.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("|Num|Expenses/Income| category| Description| Money|\n");

printf("|\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\n");

p = my\_head->first;

for (i = 0; i < my\_head->count; i++)

{

if( (p->info).expenses\_income == 1)

{

printf("|%3d| Income|%9.22s|%21.22s|%12.0f|\n", i+1, (p->info).category, (p->info).description, (p->info).money);

printf("|\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\n");

} else

{

printf("|%3d| Expense|%9.22s|%21.22s|%12.0f|\n", i+1, (p->info).category, (p->info).description, (p->info).money);

printf("|\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\n");

}

p = p->next;

}

}

char \*get\_category() // Выбор категории. Возвращает число (пункт в списка категории)

{

int q, // Выбор пункта из списка категорий

i; // Индекс категории

char \*str; // Очередная категория

for (i = 0; i < cat\_num; i++)

printf("%d. %s\n", i+1, categories[i]);

//Очистка потока ввода

do

{

q = get\_int();

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\*));

printf("Your choice %s\n", categories[q-1]);

str = (char\*)malloc(strlen(categories[q-1])\*sizeof(char));

strcpy(str, categories[q-1]);

return str;

}

void fill\_node(manager \*list, int \*bl) // Ввод очередной структуры

{

do

{

printf("Choose this income or expense. Income - 1, Expense - 2\n");

list->expenses\_income = get\_int();

if(list->expenses\_income != 1 && list->expenses\_income != 2)

puts("You entered an invalid number");

} while (list->expenses\_income <= 0 || list->expenses\_income > 2);

printf("Select the desired category: \n");

list->category = get\_category();

printf("Enter a description: \n");

list->description = get\_string(bl);

do

{

printf("Enter the amount of money: \n");

list->money = get\_float();

if(list->money <= 0)

puts("The number must be positive!");

} while (list->money <= 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;

}

new\_node->next = NULL;

my\_head->last = new\_node;

my\_head->count++;

}

}

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;

int c;

do

{

printf("1 - Add node to start\n2 - Add node to end\n3 - Insert node\nPress 0 to stop\n");

c = get\_int();

if (c != 0) p = create\_node(bl);

switch (c)

{

case 1:

add\_first(HEAD, p);

break;

case 2:

add\_last(HEAD, p);

break;

case 3:

insert(HEAD, p);

break;

case 0:

break;

default:

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

}

} while (c != 0);

}

// Копирование узла

Node \*copy\_node(Node \*NODE, int \*bl)

{

Node \*p=NULL;

p = (Node\*)malloc(sizeof(Node));

(p->info).category = (char\*)malloc(MAXLEN\*sizeof(char));

(p->info).description = (char\*)malloc(MAXLEN\*sizeof(char));

if((p->info).category!=NULL && (p->info).description!=NULL)

{

strcpy((p->info).category, (NODE->info).category);

strcpy((p->info).description, (NODE->info).description);

(p->info).expenses\_income = (NODE->info).expenses\_income;

(p->info).money = (NODE->info).money;

} 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

void clean\_node(Node \*node)

{

//if((node->info).description != NULL)

free((node->info).description);

//if((node->info).category != NULL)

free((node->info).category);

(node->info).category = NULL;

(node->info).description = NULL;

free(node);

}

// 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? (1/any)\n");

c = get\_int();

if (c == 1)

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; 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("Delete more? (1/any)\n");

c = get\_int();

}

else

c = 1;

} while ((c == 1) && my\_head->count > 0);

}

// Сравнение двух очередных элементов в списке

int compare(Node \*left, Node \*right, int type)

{

int result;

switch (type)

{

case 1:

if ((left->info).money > (right->info).money)

result = 1;

else if ((left->info).money < (right->info).money)

result = -1;

else

result = 0;

break;

case 2:

if ((left->info).expenses\_income > (right->info).expenses\_income)

result = 1;

else if ((left->info).expenses\_income < (right->info).expenses\_income)

result = -1;

else

result = 0;

break;

}

return result;

}

// Сортировка

void sort(Head \*HEAD)

{

int i, j, // Переменные в цикле

type; // Вид сортировки

int decrease; // Enter - по убыванию. Иначе по возрастанию

Node \*p=NULL, \*buff=NULL, \*temp = NULL;

do

{

printf("Choose sort number\n");

printf("1 - By price, 2 - By income/expenses\n");

type = get\_int();

} while (type<1 || type>2);

if(type == 1)

{

printf("Sort Descending? (1/any)\n");

decrease = get\_int();

}

else

{

printf("Derive expenses first? (1/any)\n");

decrease = get\_int();

}

p = HEAD->first;

for (i=1; i<=HEAD->count-1; i++)

{

buff = p->next;

for (j=i+1; j<=HEAD->count; j++)

{

if ((decrease==1) ? (compare(buff, p, type) > 0) : (compare(buff, p, type) < 0))

{

swap(HEAD, i, j);

temp = p;

p = buff;

buff = temp->next;

}

else

buff = buff->next;

}

p = p->next;

}

}

// Фильтр. Возвращает голову нового списка

Head \*selected(Head \*my\_head, int \*bl)

{

Head \*NEW\_HEAD = NULL;

Node \*p = NULL;

int i, expenses\_income;

float max\_money = 0;

char \*category = NULL;

NEW\_HEAD = make\_head(bl);

printf("Will the new list consist only of income or expenses? Income - 1, Expenses - 2\n");

expenses\_income = get\_int();

printf("The new list will consist only of this category: \n");

category = get\_category();

printf("The maximum amount. For example: if you enter 100, then amounts > 100 will not be displayed: ");

max\_money = get\_float();

p = my\_head->first;

for (i=0; i<my\_head->count; i++)

{

if (((p->info).money <= max\_money) && (strcmp((p->info).category, category)==0) && ((p->info).expenses\_income == expenses\_income))

add\_last(NEW\_HEAD, copy\_node(p, bl));

p = p->next;

}

free(category);

return NEW\_HEAD;

}

// Высвобождение памяти списка

Head \*clean\_list(Head \*HEAD)

{

Node \*p, \*temp;

int i;

p = HEAD->first;

for (i = 0; i < HEAD->count; i++)

{

temp = p;

p = p->next;

temp->next = NULL;

temp->prev = NULL;

clean\_node(temp);

}

free(HEAD);

return NULL;

}

// Редактирование узла

void edit\_node(Head \*list)

{

int change\_int,

variant1,

variant2,

exit\_flag;

float change\_float;

char \*change\_str;

Node \*temp\_node = NULL;

print\_managers(list);

do

{

puts("\nEnter number of element of list");

printf(">");

variant1 = get\_int();

if (variant1 > list->count || variant1 <= 0)

printf("Number should be greater than 0 and less than length");

}

while(variant1 > list->count || variant1 <= 0);

exit\_flag = 1;

do

{

temp\_node = list->first;

for(int i = 1; i < variant1; i++)

temp\_node = temp\_node->next;

puts("\nEdit info menu");

puts("1. Edit expenses\_income");

puts("2. Edit category");

puts("3. Edit description");

puts("4. Edit money");

puts("0. Return to back");

variant2 = get\_int();

switch (variant2)

{

case 1:

system(CLEAR);

printf("Enter expenses or income. 1 - Income 2 - Expenses: ");

change\_int = get\_int();

(temp\_node->info).expenses\_income = change\_int;

break;

case 2:

system(CLEAR);

printf("Enter category\n");

change\_str = get\_category();

free((temp\_node->info).category);

(temp\_node->info).category = change\_str;

break;

case 3:

system(CLEAR);

printf("Enter description\n");

change\_str = get\_string();

free((temp\_node->info).description);

(temp\_node->info).description = change\_str;

break;

case 4:

system(CLEAR);

printf("Enter money\n");

change\_float = get\_float();

(temp\_node->info).money = change\_float;

break;

case 0:

exit\_flag = 0;

break;

default:

printf("ERROR");

break;

}

}

while(exit\_flag);

}

// Поиск (1)

Head \*search(Head \*list, manager search\_param, int field, int\* bl)

{

Head\* search\_result = NULL;

Node\* temp\_node = NULL;

search\_result = make\_head(bl);

switch(field)

{

case 1:

for (temp\_node = list->first; temp\_node; temp\_node = temp\_node->next)

if ((temp\_node->info).expenses\_income == search\_param.expenses\_income)

{

add\_last(search\_result, copy\_node(temp\_node, bl));

}

break;

case 2:

for (temp\_node = list->first; temp\_node; temp\_node = temp\_node->next)

if (!strcmp((temp\_node->info).category, search\_param.category))

{

add\_last(search\_result, copy\_node(temp\_node, bl));

}

break;

case 3:

for (temp\_node = list->first; temp\_node; temp\_node = temp\_node->next)

if (!strcmp((temp\_node->info).description, search\_param.description))

{

add\_last(search\_result, copy\_node(temp\_node, bl));

}

break;

case 4:

for (temp\_node = list->first; temp\_node; temp\_node = temp\_node->next)

if ((temp\_node->info).money == search\_param.money)

{

add\_last(search\_result, copy\_node(temp\_node, bl));

}

break;

}

return search\_result;

}

// Поиск (2)

void search\_managers(Head \*list, int\* bl)

{

int variant,

i;

Head \*search\_list = NULL,

\*temp = NULL;

manager search\_param;

int exit\_flag;

unsigned char flags[6];

for (i = 0; i < 6; i++)

flags[i] = 1;

search\_list = list;

i = 0;

do

{

puts("1 - Expenses/income");

puts("2 - Category");

puts("3 - Description");

puts("4 - Money");

variant = get\_int();

switch (variant)

{

case 1:

if (flags[0] == 1)

{

puts("Enter expenses or income. 1 - Income 2 - Expenses");

search\_param.expenses\_income = get\_int();

flags[0] = 0;

temp = search\_list;

search\_list = search(search\_list, search\_param, 1, bl);

if(i != 0)

temp = clean\_list(temp);

i++;

}

else

printf("You re-enter");

break;

case 2:

if (flags[1] == 1)

{

puts("Enter category");

search\_param.category = get\_category();

flags[1] = 0;

temp = search\_list;

search\_list = search(search\_list, search\_param, 2, bl);

if(i != 0)

temp = clean\_list(temp);

i++;

free(search\_param.category);

}

else

printf("You re-enter");

break;

case 3:

if (flags[2] == 1)

{

puts("Enter description");

search\_param.description = get\_string();

flags[2] = 0;

temp = search\_list;

search\_list = search(search\_list, search\_param, 3, bl);

if(i != 0)

temp = clean\_list(temp);

i++;

free(search\_param.description);

}

else

printf("You re-enter");

break;

case 4:

if (flags[3] == 1)

{

puts("Enter money");

search\_param.money = get\_int();

flags[3] = 0;

temp = search\_list;

search\_list = search(search\_list, search\_param, 4, bl);

if(i != 0)

temp = clean\_list(temp);

i++;

}

else

printf("You re-enter");

break;

default:

printf("Error");

break;

}

if(i != 4)

{

puts("\nDo you want to choose more param? (1/any)");

printf(">");

exit\_flag = get\_int();

}

else

exit\_flag = 0;

}

while(exit\_flag == 1);

if (search\_list != NULL && search\_list->first != NULL)

{

print\_managers(search\_list);

pause();

}

else

{

printf("Nothing found\n");

pause();

}

if (i != 0)

search\_list = clean\_list(search\_list);

}

# 6.3 get.c

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <string.h>

#include "./include/get.h"

#include "./include/const.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 "./include/dbl\_list.h"

#include "./include/get.h"

// Высвобождение памяти для массива строк

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");

}

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++;

}

}

str\_array[m][j-k]='\0';

} 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");

free(path);

}

else df = fopen("database.csv", "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

valid\_file = 0;

return valid\_file;

}

// Запись в файл

int write\_to\_file(Head \*HEAD, int MODE)

{

FILE \*file; // Указатель на файл

int valid\_file = 1; // -1 - Файл не открылся. 0 - Файла закрылся с ошибкой. 1 - все ок.

if (MODE) // Записать в новый файл

{

char \*path;

puts("Type path to file or his name: ");

path = get\_string();

file = fopen(path, "w");

free(path);

}

else file = fopen("output.csv", "w"); // Записать в файл по умолчанию

if (file == NULL)

{

printf("Error opening file!\n");

valid\_file = -1;

}

else

{

int i;

Node \*p=NULL;

p = HEAD->first;

for (i = 0; i < HEAD->count; i++)

{

if((p->info).expenses\_income == 1)

fprintf(file, "Income;");

else

fprintf(file, "Expenses;");

fprintf(file, "%s;", (p->info).category);

fprintf(file, "%s;", (p->info).description);

fprintf(file, "%.2f", (p->info).money);

if (i < HEAD->count-1)

fprintf(file, "\n");

p = p->next;

}

if (fclose(file) == EOF)

valid\_file = 0;

}

return valid\_file;

}

# 6.5 common.c

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <string.h>

#include "./include/dbl\_list.h"

#include "./include/common.h"

#include "./include/get.h"

#include "./include/w\_file.h"

#include "./include/const.h"

// Меню

int Menu(int q)

{

int Q; // Выбор пользователя

system(CLEAR);

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.csv)");

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("6 - Edit element of list");

puts("7 - Search manager");

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.csv)");

puts("2 - Select a file");

puts("0 - Come back");

}

printf("Select menu item - ");

Q = get\_int();

printf("\n");

return Q;

}

// Справка

void Help()

{

system(CLEAR);

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");

}

// Замена к system("pause")

void pause()

{

puts("Press Enter to continue");

getchar();

}

// Работа с линейным двусвязным списком

int dbl\_list()

{

Head \*HEAD = NULL, // Голова списка

\*NEW\_HEAD = NULL; // Голова результирующего списка

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

int 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

{

printf("Do you want see list of managers? (1/any)\n");

c = get\_int();

if (c == 1)

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();

if((first<1 || second>HEAD->count) || (second<1 || first>HEAD->count))

puts("Incorrect input");

} 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? (1/any)");

c = get\_int();

} while (c == 1);

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.");

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");

pause();

break;

case 6:

edit\_node(HEAD);

break;

case 7:

search\_managers(HEAD, &bl);

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;

}

pause();

} while (Q && bl);

} else printf("Error memory");

// Высвобождение памяти

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

void pause(); // Пауза

int dbl\_list(); // Линейный двусвязный список

#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); // Ввод узла

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); // Перестановка двух элементов

void remove\_node(Head \*my\_head); // Удаление узла

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

void clean\_node(Node \*node); // Высвобождение памяти узла

Head \*clean\_list(Head \*HEAD); // Высвобождение памяти списка

void edit\_node(Head \*list); // Редактирование узла

Head \*search(Head \*list, manager search\_param, int field, int\* bl); // Поиск (1)

void search\_managers(Head \*list, int\* bl); // Поиск (2)

# 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

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

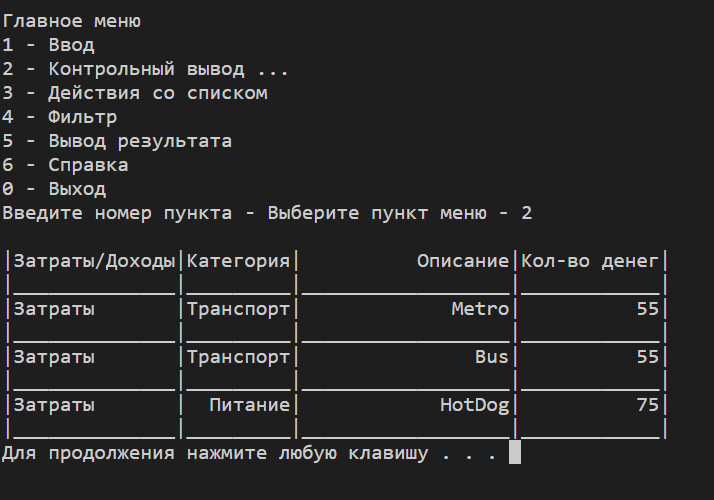
Node \*convert\_to\_node(char \*\*s2); // Конвератация массива строк в элемент списка

#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++».