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](#_Toc23608)**

**[1. Задание 3](#_1. Задание)**

**[2. Уточнение задания 4](#_2. Уточнение задания)**

**[3. Контрольные примеры 8](#_3. Контрольные примеры)**

**[4. Краткое описание алгоритма 9](#__x000F_4. Краткое описание алгоритма)**

**[5. Структура вызова функций 10](#_5. Структура вызова функций)**

**[6. Функции 11](#__x0001__x000F_6. Функции)**

[6.1 Функция main 1](#_Toc16158)1

[6.2 Функция get\_string 1](#_Toc5658)7

[6.3 Функция get\_int 1](#_Toc17485)8

[6.4 Функция get\_float 1](#_Toc17073)9

[6.5 Функция get\_subject 2](#_Toc1338)0

[6.6 Функция fill\_node 2](#_Toc14962)1

[6.7 Функция make\_head 2](#_Toc17028)2

[6.8 Функция add\_item 2](#_6.8 Функция add_item)3

[6.9 Функция create\_node 2](#_Toc6285)4

[6.10 Функция add\_first 2](#_Toc23393)5

[6.11 Функция add\_last 2](#_Toc21767)6

[6.12 Функция insert 2](#_Toc12003)7

[6.13 Функция swap 2](#_Toc25949)8

[6.14 Функция remove\_node 2](#_Toc27765)9

[6.15 Функция copy\_node](#_Toc13194) 30

[6.16 Функция clean\_node](#_Toc30050) 31

[6.17 Функция clean\_list 3](#_Toc9729)2

[6.18 Функция print\_tutors 3](#_Toc16221)3

[6.19 Функция selected 3](#_Toc2191)4

[6.20 Функция sort 3](#_Toc26713)5

[6.21 Функция compare 3](#_Toc25929)6

[6.22 Функция Menu 3](#_Toc28226)7

[6.23 Функция get\_database 38](#_6.24 Функция write_to_file)

[6.24 Функция write\_to\_file 39](#_6.24 Функция write_to_file)

[6.25 Функция simple\_split 40](#_6.25 Функция simple_split)

[6.26 Функция covert\_to\_node 41](#_6.26 Функция convert_to_node)

**[7. Текст программы](#_6. Текст программы) 42**

**[8. Пример работы программы 5](#_Toc20814)6**

[8.1. Вывод программы 5](#_Toc24594)7

[8.2 Результат работы программы 5](#_Toc21134)9

**[Заключение](#_Заключение) 59**

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. Гарантируется заведомо правильно оформленный входной файл, правильным считается файл, где каждое поле непустое, отделено символом «точка с запятой» и отсутствуют пустые строки

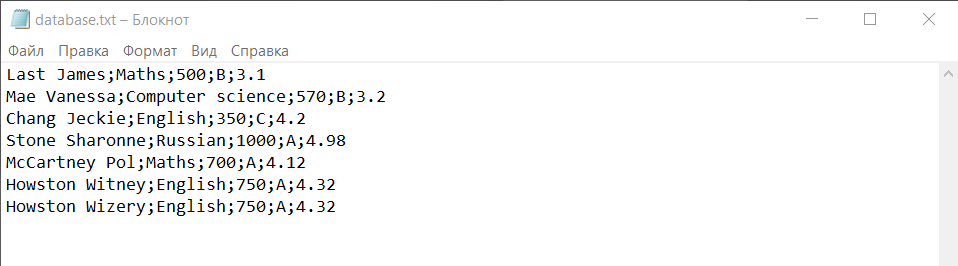


Рис.1. Пример правильно оформленного входного файла

1. 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 пользователь может ввести исходные 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 ещё один в начало, середину или конец списка. Поля структур вводятся до тех пор, пока пользователь не захочет прекратить ввод. Таким образом формируется линейный двусвязный список.

При выборе пункта 2 происходит вывод введенных пользователем полей структур в виде таблицы.

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

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

1. Учебная дисциплина
2. Максимально допустимая цена репетитора в час
3. Минимальный рейтинг репетитора

При выборе пункта 5 пользователю даётся на выбор загрузить новый список в файл или вывести в терминал. Если не было введено структур, удовлетворяющим трём 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. Структура вызова функций

# struct_main_1 (1)

Рисунок 2. Структура вызова функций

# 6. Функции

## 6.1 Функция main

**Назначение:**

Является точкой входа в программу.

**Описание переменных:**

Описание переменных приведено в таблице 2.

Таблица 2. Описание переменных функции main

|  |  |  |
| --- | --- | --- |
| Имя переменной | Тип | Назначение |
| Q, Q1, Q5, Q52, Q12, Q3, output | int | Организация меню и подменю |
| valid\_file | int | Проверки валидности файлов |
| first, second, buff | int | Выбор различных элементов списка |
| p | Node\* | Указатель на новую структуру |
| output | int | Проверка проведённой фильтрации списка |
| HEAD | Head\* | Указатель на голову списка |
| NEW\_HEAD | Head\* | Указатель на голову сформированного списка |

**Схема алгоритма:**

Схема алгоритма для функции main представлена на рисунке 3.

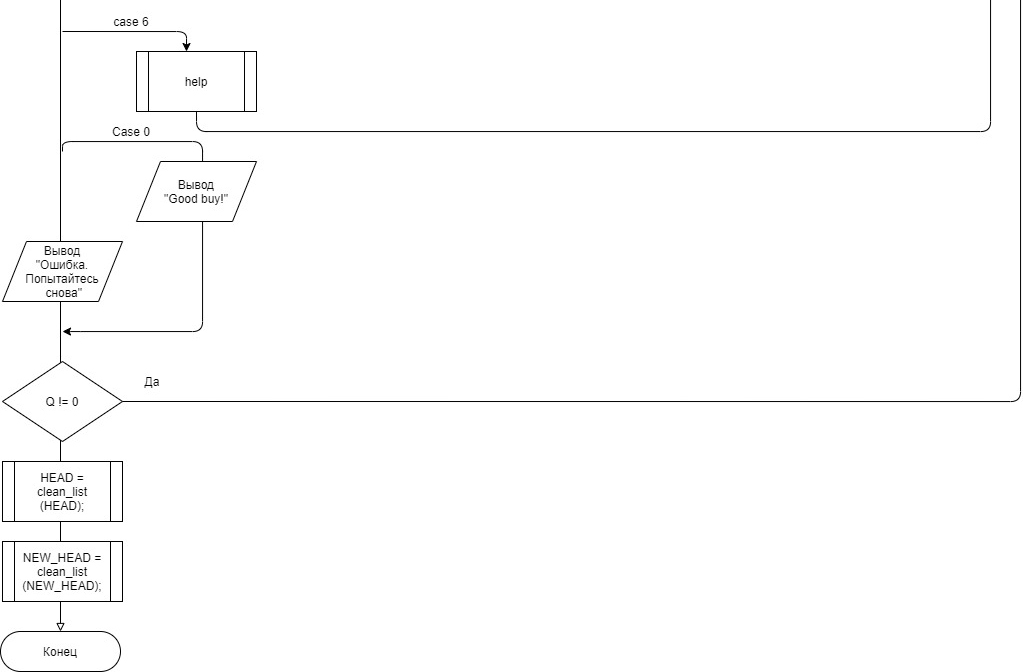
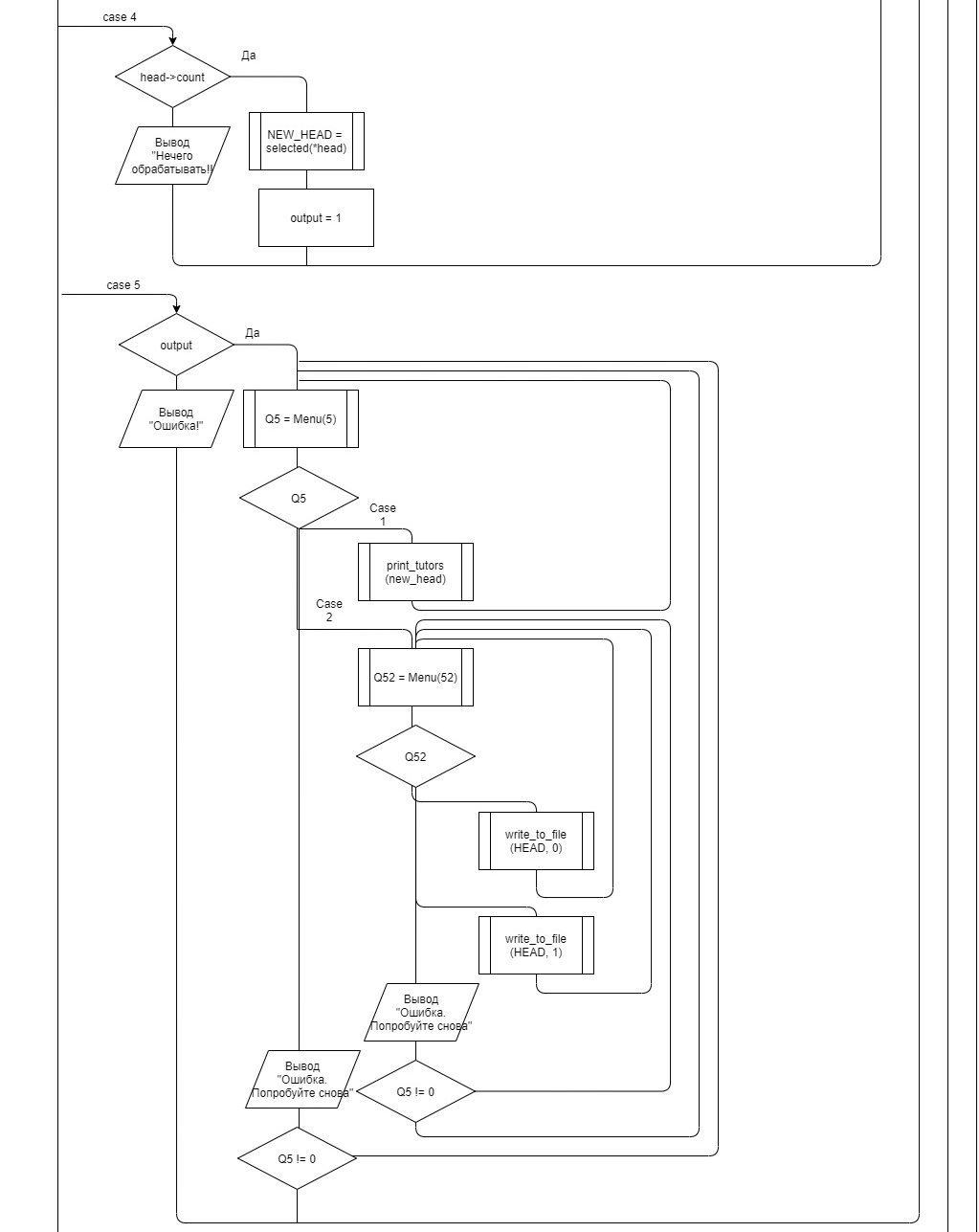
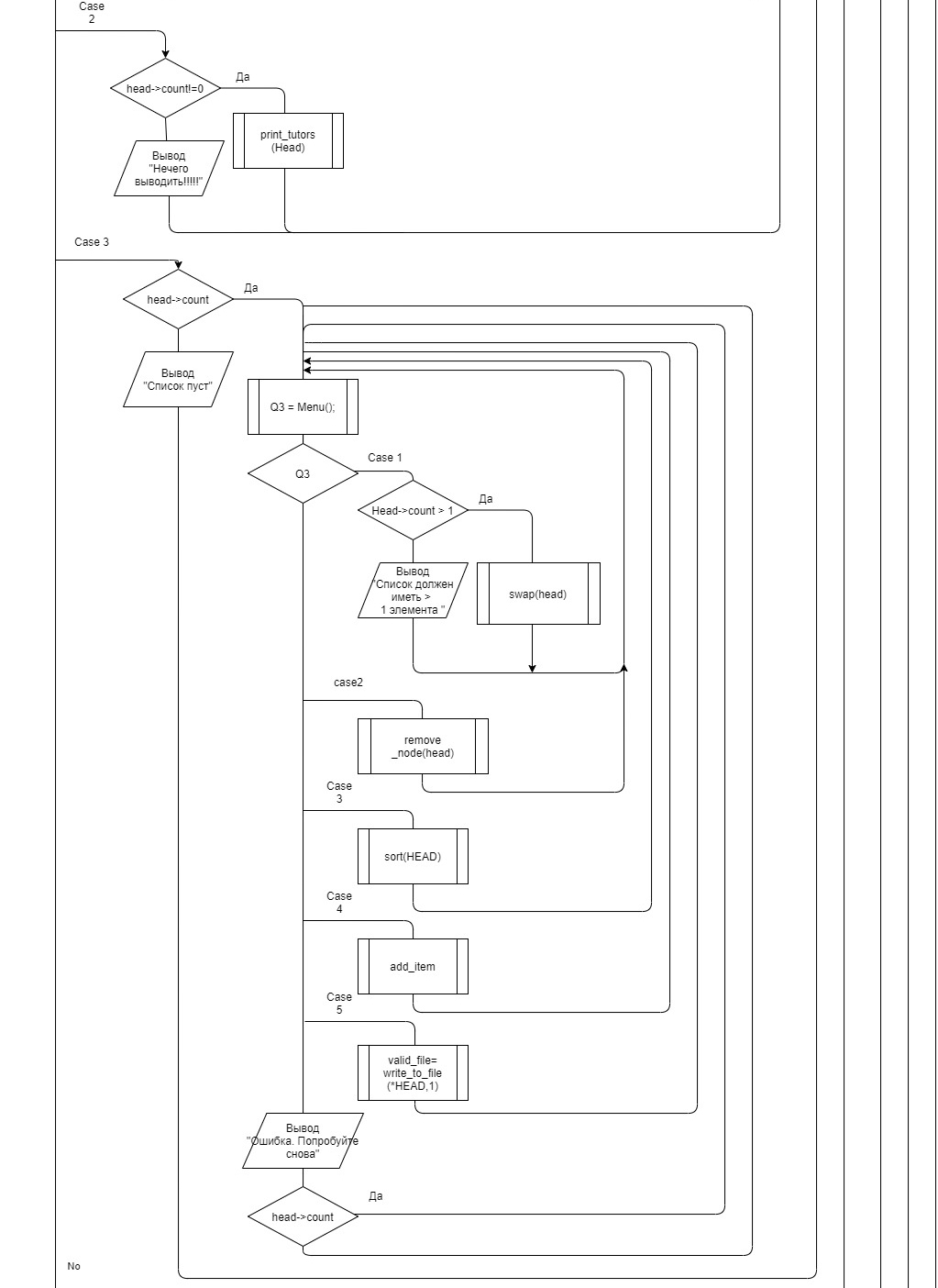
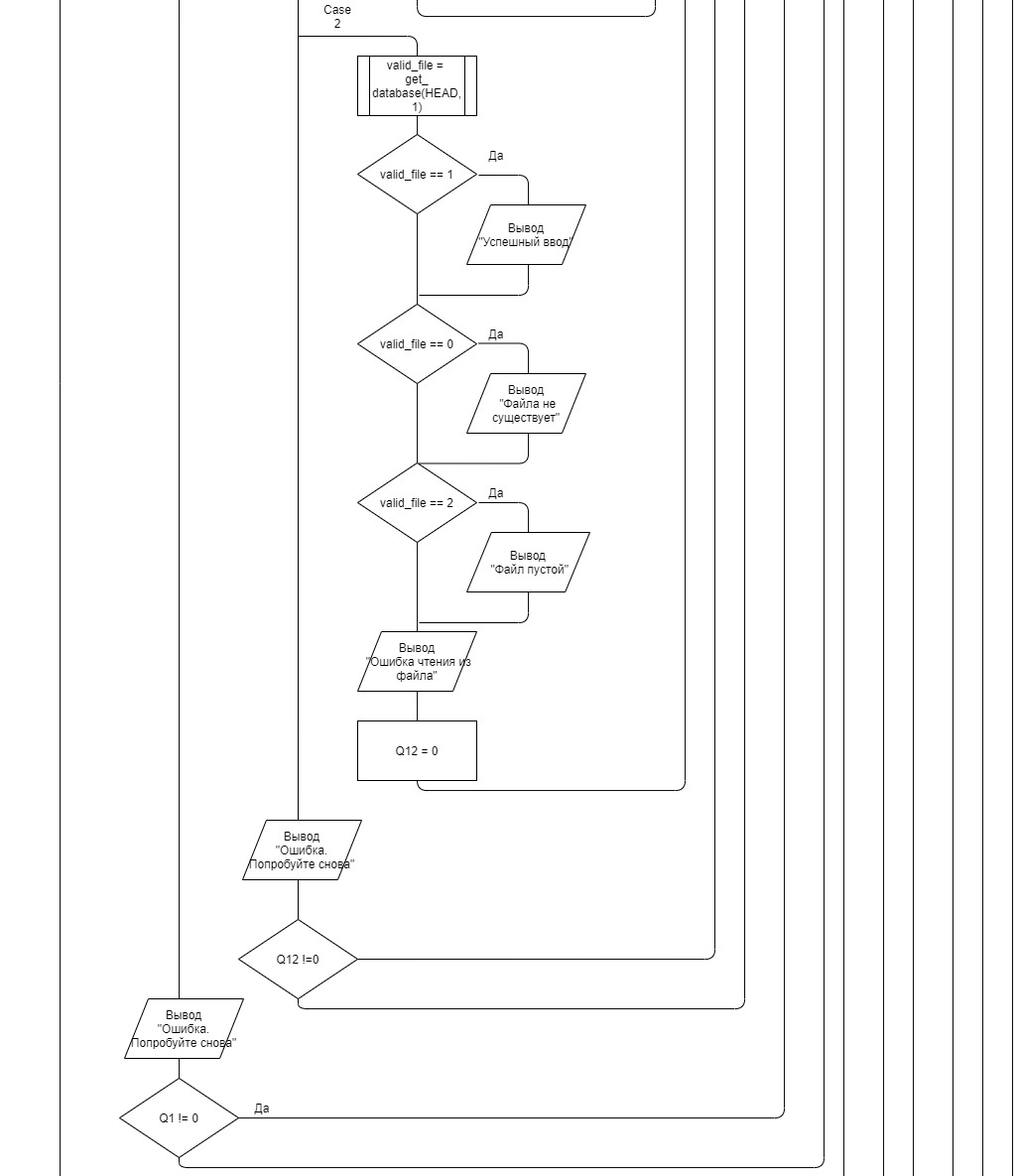
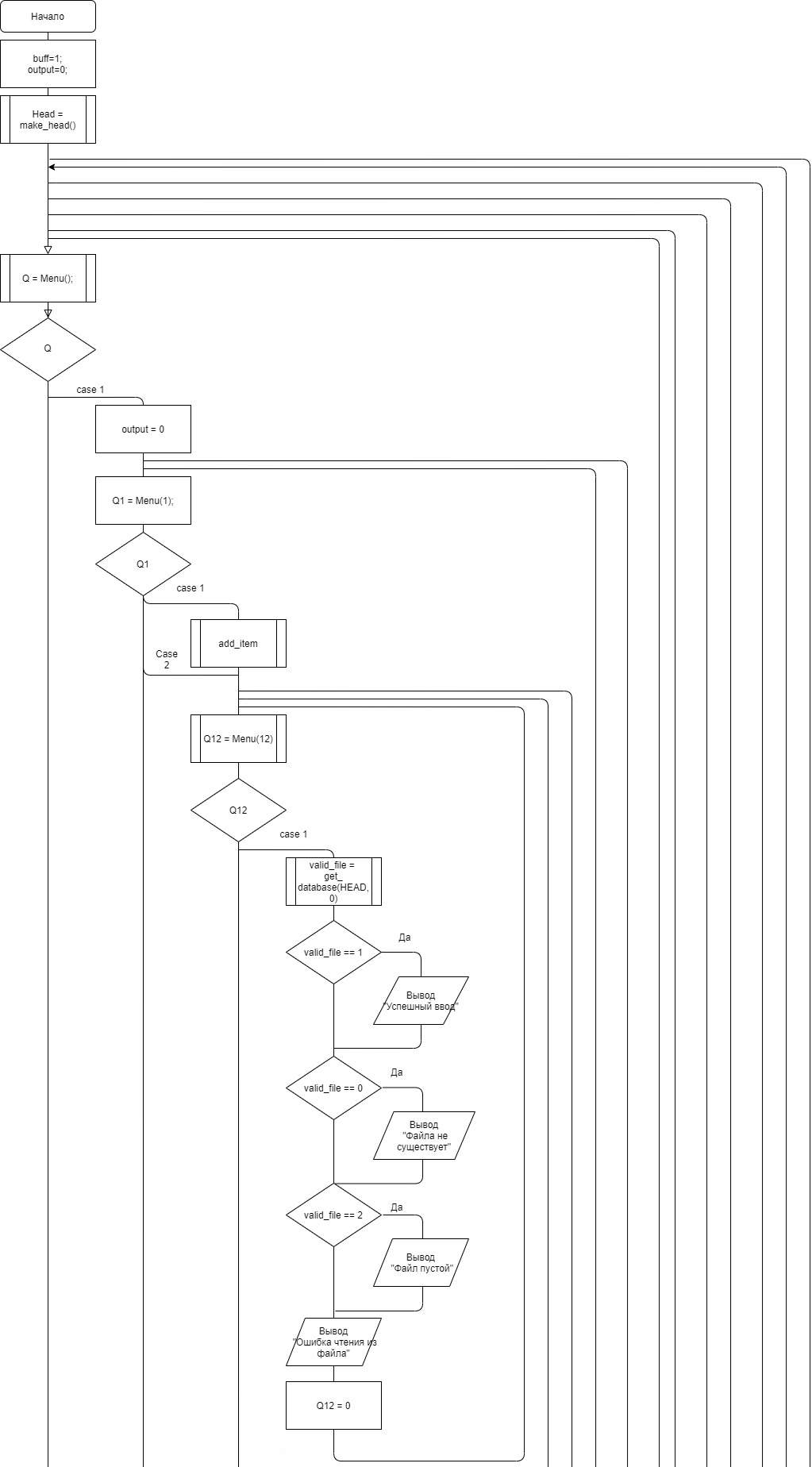


Рис.4 Схема функции main

## 6.2 Функция get\_string

**Назначение:**

Функция ввода строки.

**Прототип:**

char \*get\_string();

**Пример вызова:**

list->name = get\_string();

**Описание переменных:**

Описание переменных приведено в таблице 3.

Таблица 3. Описание переменных функции get\_string

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | c | char | Символ строки |
| string | сhar\* | Адрес начала строки |
| i | int | Индекс символа строки |

## 6.3 Функция get\_int

**Назначение:**

Функция ввода числа типа int.

**Прототип:**

int get\_int(void);

**Пример вызова:**

list->price = get\_int();

**Описание переменных:**

Описание переменных приведено в таблице 4.

Таблица 4. Описание переменных функции get\_int

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | line | char | Буфер |
| curChar | сhar | Переменная для определения корректности ввода |
| temp | int | Проверка успешной записи числа |
| result | int | Сформированное значение типа int |

## 6.4 Функция get\_float

**Назначение:**

Функция ввода числа типа float.

**Прототип:**

int get\_float(void);

**Пример вызова:**

list->rating = get\_float();

**Описание переменных:**

Описание переменных приведено в таблице 5.

Таблица 5. Описание переменных функции get\_float

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | line | char | Буфер |
| curChar | char | Переменная для определения корректности ввода |
| temp | int | Проверка успешной записи числа |
| result | float | Сформированное значение типа float |

## 6.5 Функция get\_subject

**Назначение:**

Функция для выбора категории пользователем.

**Прототип:**

сhar \*get\_subject(void);

**Пример вызова:**

List->subject = get\_subject();

**Описание переменных:**

Описание переменных приведено в таблице 6.

Таблица 6. Описание переменных функции get\_subject

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | choice | char\* | Указатель на выбранный пользователем предмет |
| q | int | Номер выбранного предмета |
| i | int | Счетчик цикла |

## 6.6 Функция fill\_node

**Назначение:**

Заполнение структуры.

**Прототип:**

void \*add\_item(tutor\*);

**Пример вызова:**

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

**Описание переменных:**

Описание переменных приведено в таблице 7.

Таблица 7. Описание переменных функции add\_item

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | list | tutor\* | Указатель на структуру |

## 6.7 Функция make\_head

**Назначение:**

Инициализация головы cписка.

**Прототип:**

Head \*make\_head();

**Пример вызова:**

NEW\_HEAD = make\_head();

**Описание переменных:**

Описание переменных приведено в таблице 8.

Таблица 8. Описание переменных функции make\_head

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | ph | Head\* | Указатель на голову списка |

## 

## 6.8 Функция add\_item

**Назначение:**

Добавление узла в список на выбранное место(начало, конец, произвольно).

**Прототип:**

voidadd\_item(\*Head);

**Пример вызова:**

add\_item(\*HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 9.

Таблица 9. Описание переменных функции add\_item

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на структуру головы |
| Локальная | new\_node | Node\* | Указатель на адрес элемента списка |
| Локальная | buff | int | Организация выбора |

## 6.9 Функция create\_node

**Назначение:**

Создание элемента списка.

**Прототип:**

Node \*create\_node()

**Пример вызова:**

p = create\_node();

**Описание переменных:**

Описание переменных приведено в таблице 10.

Таблица 10. Описание переменных функции сreate\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | new\_node | Node\* | Указатель на адрес элемента списка |

**6.10 Функция add\_first**

**Назначение:**

Добавить элемент в начало списка.

**Прототип:**

void \*add\_first(Head\*, Node\*);

**Пример вызова:**

add\_first(my\_head, new\_node);

**Описание переменных:**

Описание переменных приведено в таблице 11.

Таблица 11. Описание переменных функции add\_first

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | new\_node | Node\* | Указатель на новый узел |
| my\_head | Head\* | Указатель на адрес головы списка |

**6.11 Функция add\_last**

**Назначение:**

Добавить элемент в конец списка.

**Прототип:**

void \*add\_last(Head\*, Node\*);

**Пример вызова:**

add\_last(my\_head, new\_node);

**Описание переменных:**

Описание переменных приведено в таблице 12.

Таблица 12. Описание переменных функции add\_last

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | new\_node | Node\* | Указатель на новый узел |
| my\_head | Head\* | Указатель на адрес головы списка |

**6.12 Функция insert**

**Назначение:**

Вставка элемента в произвольное место списка.

**Прототип:**

void insert(Head\*, Node\*);

**Пример вызова:**

insert(HEAD, p);

**Описание переменных:**

Описание переменных приведено в таблице 13.

Таблица 13. Описание переменных функции insert

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | new\_node | Node\* | Указатель на новый узел |
| my\_head | Head\* | Указатель на адрес головы списка |
| Локальная | i | int | Счетчик для цикла |
| pos | int | Номер позиции вставки |
| p | Node\* | Указатель на узел |

**6.13 Функция swap**

**Назначение:**

Перестановка двух элементов в списке.

**Прототип:**

void swap(Head \*HEAD);

**Пример вызова:**

swap(HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 14.

Таблица 14. Описание переменных функции swap

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на голову списка |
| Локальная | p-one | Node\* | Указатель на узел |
| p-two | Node\* | Указатель на узел |
| buff-one | Node\* | Адрес первого элемента |
| buff-two | Node\* | Адрес второго элемента |
| first | int | Номер позиции первого элемента |
| second | int | Номер позиции второго элемента |

**6.14 Функция remove\_node**

**Назначение:**

Удалить выбранный элемент из списка.

**Прототип:**

void remove\_node(Head\*)

**Пример вызова:**

remove\_node(HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 15.

Таблица 15. Описание переменных функции remove\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | my\_head | Head\* | Указатель на голову |
| Локальная | i | int | Счетчик для цикла |
| pos | int | Номер позиции элемента, который необходимо удалить |
| buff | int | Флаг организации работы |
| p | Node\* | Указатель на узел |

**6.15 Функция copy\_node**

**Назначение:**

Копирование элемента списка.

**Прототип:**

Node \*copy\_node(Node\*);

**Пример вызова:**

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

**Описание переменных:**

Описание переменных приведено в таблице 16.

Таблица 16. Описание переменных функции copy\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | NODE | Node\* | Указатель на адрес элемента списка |
| Локальная | p | Node\* | Указатель на узел |

**6.16 Функция clean\_node**

**Назначение:**

Очистка памяти для одной записи.

**Прототип:**

Node \*clean\_node(Node\*);

**Пример вызова:**

clean\_node(p);

**Описание переменных:**

Описание переменных приведено в таблице 17.

Таблица 17. Описание переменных функции clean\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | node | Node\* | Указатель на адрес удаляемого элемента |

**6.17 Функция clean\_list**

**Назначение:**

Очистка памяти под список.

**Прототип:**

Head \*clean\_list(Head\*);

**Пример вызова:**

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

**Описание переменных:**

Описание переменных приведено в таблице 18.

Таблица 18. Описание переменных функции clean\_list

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на голову |
| Локальная | p | Node\* | Адрес первого элемента списка |
| temp | Node\* | Временная переменная |
| i | int | Счетчик для цикла |

## 6.18 Функция print\_tutors

**Назначение:**

Печать списка в виде таблицы.

**Прототип:**

void print\_tutors(tutor\*, int);

**Пример вызова:**

print\_tutors(data, k);

**Описание переменных:**

Описание переменных приведено в таблице 19.

Таблица 19. Описание переменных функции print\_tutors

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | i | int | Счетчик для цикла |
| p | Node\* | Хранит текущий элемент списка |
| Формальная | my\_head | Head\* | Указатель на голову |

## 6.19 Функция selected

**Назначение:**

Функция для сортировки структур по заданным критериям.

**Прототип:**

Head \*selected(Head\*);

**Пример вызова:**

NEW\_HEAD = selected(HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 20.

Таблица 20. Описание переменных функции selected

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | NEW\_HEAD | Head\* | Указатель на новую голову |
| i | int | Счетчик цикла |
| maxPrice | int | Критерий суммы |
| minRating | float | Kритерий рейтинга |
| p | Node\* | Временная переменная |
| subject | char\* | Указатель на критерий предмета |
| Формальная | my\_head | Head\* | Указатель на голову |

**6.20 Функция sort**

**Назначение:**

Сортировка списка по произвольному полю.

**Прототип:**

void sort(Head \*HEAD);

**Пример вызова:**

sort(HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 21.

Таблица 21. Описание переменных функции sort

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на голову списка |
| Локальная | i | int | Счетчик для списка |
| j | int | Счетчик для списка |
| decrease | int | Флаг сортировки по возрастанию/убыванию |
| p | Node\* | Адрес второго элемента |
| buff | Node\* | Буфер хранящий указатель на структуру |
| temp | Node\* | Временная переменная |
| type | int | Тип поля, по которому сортируется список |

## 6.21 Функция compare

**Назначение:**

Сравнение двух элементов списка.

**Прототип:**

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

**Пример вызова:**

compare(buff, p, type);

**Описание переменных:**

Описание переменных приведено в таблице 22.

Таблица 22. Описание переменных функции compare

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | result | int | Результат сравнения |
| Формальная | right | Node\* | «Правый» элемент сравнения |
| left | Node\* | «Левый» элемент сравнения |
| type | int | Тип поля, по которому сортируется список |

## 6.22 Функция Menu

**Назначение:**

Функция вывода меню.

**Прототип:**

int Menu();

**Пример вызова:**

Q **=** Menu();

**Описание переменных:**

Описание переменных приведено в таблице 23.

Таблица 23. Описание переменных функции Menu

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | Q | int | Номер пункта меню, выбираемый пользователем |
| Формальная | q | int | Уровень меню/подменю |

6.23 Функция get\_database

**Назначение:**

Чтение из файла и последующее формирование списка

**Прототип:**

int get\_database(Head\*, int);

**Пример вызова:**

valid\_file = get\_database(HEAD, 1);

**Описание переменных:**

Описание переменных приведено в таблице 24.

Таблица 24. Описание переменных функции get\_database

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на голову списка |
| MODE | int | Режим работы(ввод из файла по умолчанию/произвольный файл) |
| Локальная | p | Node\* | Указатель на новый элемент списка |
| slen, i, count, flag | int | Организация исполнения алгоритма |
| valid\_file | int | Код ошибки или же успешного чтения из файла |
| sep | char | Символ разделителя полей у элемента |
| s1[MAXSTR] | char | Временная строка, хранящая весь элемент списка |
| s2 | char\*\* | Указатель на вектор полей структуры |
| df | FILE\* | Используемый файл |

# 6.24 Функция write\_to\_file

**Назначение:**

Запись списка 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 25.

Таблица 25. Описание переменных функции write\_to\_file

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на голову списка |
| MODE | int | Режим работы(ввод из файла по умолчанию/произвольный файл) |
| Локальная | valid\_file | int | Код ошибки или же успешного чтения из файла |
| df | FILE\* | Используемый файл |

# 6.25 Функция simple\_split

**Назначение:**

Разделение строки файла на массив строк, содержащих каждое из полей будущей структуры

**Прототип:**

char \*\*simple\_split(char \*str, int length, char sep);

**Пример вызова:**

s2=simple\_split(s1,slen,sep);

**Описание переменных:**

Описание переменных приведено в таблице 26.

Таблица 26. Описание переменных функции simple\_split

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | str | Char\* | Указатель на голову списка |
| length | int | Режим работы(ввод из файла по умолчанию/произвольный файл) |
| sep | char | Разделитель строки |
| Локальная | Str\_array | Char\*\* | Указатель на вектор строк |
| i,j,k,m | int | Организация циклов |
| key,count | int | Организация работы алгоритма |

# 6.26 Функция convert\_to\_node

**Назначение:**

Разделение строки файла на массив строк, содержащих каждое из полей будущей структуры

**Прототип:**

Node \*convert\_to\_node(char \*\*s2);

**Пример вызова:**

p = convert\_to\_node(s2);

**Описание переменных:**

Описание переменных приведено в таблице 27.

Таблица 27. Описание переменных функции convert\_to\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | s2 | Char\*\* | Указатель на вектор строк |
| Локальная | p | Node\* | Указатель на стуктуру |
| len1, len2 | int | Длины строковых полей элемента списка |

# Текст программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <string.h>

#include <windows.h>

#define MAXLEN 24

#define MAXSTR 128

#ifdef \_\_linux\_\_

#define CLEAR "clear"

#else

#define CLEAR "cls"

#endif

const char \*subjects[] = {"Maths" , "Computer science", "English", "Russian", "Physics"};

typedef struct

{

// Описание полей

char \*name; // Имя

char \*subject; // Предмет

int price; // Цена за час

char qual; // Квалификация

float rating; // рейтинг преподавателя

} tutor; // Информация о репетиторе

typedef struct tutor\_elem

{

tutor info;

struct tutor\_elem \*prev;

struct tutor\_elem \*next;

} Node; // Элемент списка

typedef struct

{

int count;

Node \*first;

Node \*last;

} Head; // Голова двусвязного списка

// Кросплатформенность

void pause(); // Замена функции системной пазуы

void clean\_stdin(); // Замена функции очистки входного потока (fflush)

// Получение валидных значений //

char \*get\_string(); // Получение строки

int get\_int(); // Получение целого числа

float get\_float(); // ПОлучение числа с плавающей запятой

char \*get\_subject(); // Получение учебного предмета// Работа со списком //

void fill\_node(tutor \*list); // Заполнение информационных полей у репетитора

Head \*make\_head(); // Инициализация головы списка

void add\_item(Head \*HEAD); // Добавить элемент в список

Node \*create\_node(); // Создать элемент списка

void \*add\_first(Head \*my\_head, Node \*new\_node); // Добавить элемент в начало списка

void \*add\_last(Head \*my\_head, Node \*new\_node); // Добавить элемент в конец списка

void insert(Head \*my\_head, Node \*new\_node); // Вставка в произвольное место

void swap(Head \*HEAD, int first, int second); // Перестановка двух элементов

void sort(Head \*HEAD); // Сортировка списка по одному из полей

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

Node \*copy\_node(Node \*NODE); // Копирование элемента

void print\_tutors(Head \*my\_head); // Печать списка в виде таблицы

Head \*selected(Head \*my\_head); // Процесс фильтрации

Node \*clean\_node(Node \*); // Очистка памяти для одной записи

Head \*clean\_list(Head \*HEAD); // Очистить память под список// Работа с файлом

void clear\_str\_array(char \*\*str, int n); // Очистка вспомогательного массива строк

char \*\*simple\_split(char \*str, int length, char sep); // Преобразование строки файла в массив из полей структуры

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

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

Node \*convert\_to\_node(char \*\*s2); // Заполнение структуры при помощи вспомогательного массива строк из simple\_split

int Menu();

void Help();

int main(){

Head \*HEAD = NULL, \*NEW\_HEAD = NULL; // Голова исходного и результирующего списков

Node \*p = NULL; // Новый элемент списка

HEAD = make\_head();

int Q, Q1, Q5, Q52, Q12, Q3, output = 0; // Переменные для организации меню

int first, second, buff=1, // Выбор элемента списка

valid\_file; // Проверки валидности файлов

do {

Q = Menu(0);

switch (Q)

{

case 1: //input

output = 0;

if (HEAD->count) // Если перезаписываем список

{

HEAD = clean\_list(HEAD);

HEAD = make\_head();

if (NEW\_HEAD)

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

}

do

{

Q1 = Menu(1);

switch (Q1)

{

case 1:

add\_item(HEAD);

printf("Successful input.\n");

Q1=0;

break;

case 2:

do

{

Q12 = Menu(12);

switch (Q12)

{

case 1:

valid\_file = get\_database(HEAD, 0);

if (valid\_file==1)

printf("Successful input.\n");

else if (valid\_file == 0)

printf("Error: Nonexistent file.\n");

else if (valid\_file == 2)

printf("Error, file is empty.\n");

else

printf("Error reading from file.\n");

Q12 = 0;

break;

case 2:

valid\_file = get\_database(HEAD, 1);

if (valid\_file==1)

printf("Successful input.\n");

else if (valid\_file == 0)

printf("Error: Nonexistent 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);

Q1=0;

break;

case 0:

break;

default:

puts("Try again.");

}

} while(Q1 != 0);

break;

case 2: //output

if (HEAD->count)

print\_tutors(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(CLEAR);

printf("Do you want see list of tutors? Enter 1 if you want or enter any number otherwise\n");

buff = get\_int();

if (buff == 1)

print\_tutors(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? Enter 1 - Yes, enter any number - No");

buff = get\_int();

} while (buff == 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);

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 0:

Q3=0;

break;

default:

puts("Try again!");

break;

}

} while (Q3 != 0 && HEAD->count);

}

else

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

break;

case 4: //process

if (HEAD->count)

{

NEW\_HEAD = selected(HEAD);

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\_tutors(NEW\_HEAD);

else printf("Tutors 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("Tutors 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("Tutors 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);

Q5=0;

break;

case 0:

break;

default:

puts("Try again.");

}

} while(Q5 != 0);

else

printf("No processed data to output!\n");

break;

case 6: //help

Help();

break;

case 0: //exit

printf("Good buy, see you soon!\n");

break;

default:

printf("Error! Try again!\n");

break;

}

pause();

} while (Q);

// Очистка всего мусора

if (HEAD->count != 0)

HEAD = clean\_list(HEAD);

if (NEW\_HEAD)

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

return 0;

}

// Замена системного ожидания

void pause(){

puts("\nPress Enter to continue");

getchar();}

// Замена функции очистки входного потока (fflush)

void clean\_stdin(){

char c;

while ( scanf("%c", &c) == 1 && c != '\n');}

// Очистка массива строк

void clear\_str\_array(char \*\*str, int n){

int i; // Номер элемента списка

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

{

free(str[i]);

str[i]=NULL;

}

free(str);

str=NULL;}

// Разделение строки файла на поля элементов

char \*\*simple\_split(char \*str, int length, char sep){

char \*\*str\_array=NULL; // Указатель на вектор строк

int i,j,k,m; // Организация циклов

int key,count; // Организация работы алгоритма

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 из файла

int get\_database(Head \*HEAD, int MODE){

Node \*p; // Новый элемент списка

int slen,i,count,flag=1, // Организация исполнения алгоритма

valid\_file; // Код ошибки или же успешного чтения из файла: 1-успешно, 0-несуществующий файл, -1-ошибка чтения файла, 2-пустой файл

char sep; // Символ разделителя полей у элемента

char s1[MAXSTR]; // Временная строка, хранящая весь элемент списка

char \*\*s2=NULL; // Указатель на вектор полей структуры

FILE \*df; // Используемый файл

if (MODE)

{

char \*path;

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

path = get\_string();

df = fopen(path, "r");

free(path);

}

else df = fopen("database.txt", "r");

if(df!=NULL)

{

sep=';';

count=0;

i=0;

while(fgets(s1,MAXSTR,df) != NULL && flag)

{

slen=strlen(s1);

s1[slen-1]='\0';

slen=strlen(s1);

s2=simple\_split(s1,slen,sep);

if(s2!=NULL)

{

p = convert\_to\_node(s2);

add\_last(HEAD, p);

count++;

i++;

clear\_str\_array(s2,5);

}

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 структуру, с помощью которой будет производиться запись в файл

int valid\_file = 1; // Возвращаемый код функции

if (MODE)

{

char \*path;

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

path = get\_string();

file = fopen(path, "w");

free(path);

}

else file = fopen("output.txt", "w");

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++)

{

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

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

fprintf(file, "%d;", (p->info).price);

fprintf(file, "%c;", (p->info).qual);

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

if (i<HEAD->count-1)

fprintf(file, "\n");

p = p->next;

}

if (fclose(file) == EOF)

valid\_file = 0;

}

return valid\_file;}

// Преобразования массива строк в поля структуры

Node \*convert\_to\_node(char \*\*s2){

Node \*p = NULL; // Новый элемент списка

int len1, len2; // Длины строковых полей элемента списка

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

p->next = NULL;

p->prev = NULL;

len1=strlen(s2[0]);

len2=strlen(s2[1]);

(p->info).name=(char\*)malloc((len1+1)\*sizeof(char));

(p->info).subject=(char\*)malloc((len2+1)\*sizeof(char));

if(((p->info).name!=NULL)&&((p->info).subject!=NULL))

{

strcpy((p->info).name,s2[0]);

strcpy((p->info).subject,s2[1]);

(p->info).price=atoi(s2[2]);

(p->info).qual=s2[3][0];

(p->info).rating=atof(s2[4]);

}

else

{

puts("Out if memory! Program terminated");

p = NULL;

}

return p;}char \*get\_string() // Функция получения строки{

char c, \*string = (char\*)malloc(MAXLEN\*sizeof(char));

int i=0;

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;}

// Получение целого числа

int get\_int() // Функция получения целого числа{

char line[MAXLEN]; //buffer

char curChar = ' ';

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); //not a number actually

if (line[MAXLEN - 1] != '\n') //clear all iput

while ((curChar = getchar()) != '\n' && curChar != EOF);

return result;}

float get\_float() // Функция получения числа с плавающей запятой{

char line[MAXLEN]; //buffer

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); //not a number actually

if (line[MAXLEN - 1] != '\n') //clear all iput

while ((curChar = getchar()) != '\n' && curChar != EOF);

return result;}

char \*get\_subject() // Функция выбора Учебного предмета из заданных заранее{

char \*choice = NULL; // Указатель на строку выбранного предмета

int q, i; // Организация цикла

for (i=0; i<(int)sizeof(subjects)/sizeof(char\*); i++)

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

do

{

q = get\_int();

if (q <= 0 || q > (int)sizeof(subjects)/sizeof(char\*))

printf("This item doesn't exist. Try again.\n");

} while (q <= 0 || q > (int)sizeof(subjects)/sizeof(char\*));

choice = (char\*)malloc(MAXLEN\*sizeof(char));

strcpy(choice, subjects[q-1]);

printf("Your chosen %s\n", choice);

return choice;}

void fill\_node(tutor \*list) // Добавить элемент в список{

system(CLEAR);

list->name = (char\*)malloc(MAXLEN\*sizeof(char));

list->subject = (char\*)malloc(MAXLEN\*sizeof(char));

if (list->subject && list->name)

{

printf("Enter tutor's name: \n");

list->name = get\_string();

printf("Enter academic subject: \n");

list->subject = get\_subject();

printf("Enter price per 1 hour(RUB): \n");

list->price = get\_int();

printf("Enter qualifications: \n");

scanf("%c", &(list->qual));

clean\_stdin();

do

{

printf("Enter tutor's rating: \n");

list->rating = get\_float();

} while (list->rating <= 0 || list->rating > 5);

}}

Head \*make\_head() // Инициализация головы

{

Head \*ph=NULL; // Указатель на "голову" списка

ph=(Head\*)malloc(sizeof(Head));

if(ph!=NULL)

{

ph->count=0;

ph->first=NULL;

ph->last=NULL;

}

return ph;

}

void add\_item(Head \*HEAD) // Добавление узла в список{

Node \*p=NULL; // Указатель на новый элемент списка

int buff; // Организация выборa

do

{

printf("Enter 1 - Add node to start, 2 - Add node to end, 3 - Insert node\nEnter 0 to stop\n");

buff = get\_int();

if (buff != 0) p = create\_node();

switch (buff)

{

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 (buff != 0);}Node \*create\_node() // Создать узел списка

{

Node \*new\_node=NULL; // Указатель на новый элемент списка

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

if(new\_node)

{

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

}

new\_node->prev = NULL;

new\_node->next = NULL;

return new\_node;

}

// Добавление в начало списка

void \*add\_first(Head \*my\_head, Node \*new\_node) // Добавить новую запись первой в список

{

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++;

}

}

// Добавление в конец списка

void \*add\_last(Head \*my\_head, Node \*new\_node) // Добавить новую запись последней в список

{

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++;

}

}

// Добавление в произвольное место

void insert(Head \*my\_head, Node \*new\_node) // Вставка в любое место

{

int i, // Организация цикла

pos; // Выбор позиции

Node \*p; // Указатель на новый элемент списка

do

{

printf("What position? [From 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)) // Проверка на пустой список

{

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++) { // Проматываем до позиции перед вставляемым элементом

p = p->next;

}

new\_node->prev = p;

new\_node->next = p->next;

(p->next)->prev = new\_node;

p->next = new\_node;

my\_head->count++;

}

}

}

}

// Добавление в начало списка

void remove\_node(Head \*my\_head){

Node \*p; // Указатель на новый элемент списка

int i, pos; // Организация цикла и выбор позиции

int buff=1; // Флаг организации работы

do

{

system(CLEAR);

printf("Do you want see list of tutors? Enter 1 if you want or enter any number otherwise\n");

buff = get\_int();

if (buff == 1)

print\_tutors(my\_head);

do

{

printf("Remove element with number [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++) // Проматываем до удаляемого элемента

{

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? Enter 0 - No, any number - Yes\n");

buff = get\_int();

}

else

buff=0;

} while (buff != 0 && my\_head->count>0);}

// Поменять элементы местами

void swap(Head \*HEAD, int first, int second){

int i; // Цикл

Node \*p\_one, \*p\_two; // Указатели на выбранные стрктуры

Node \*buff\_one, \*buff\_two; // Временные указатели на новые элементы списка

p\_one = HEAD->first;

for (i=1; i<first-1; i++)

p\_one = p\_one->next; // Получим адрес предыдущего элемента

p\_two = HEAD->first;

for (i=1; i<second-1; i++) // Получим адрес предыдущего элемента

p\_two = p\_two->next;

if (first != 1)

{

buff\_one = p\_one->next; // Адрес первого элемента

buff\_two = p\_two->next; // Адрес второго элемента

p\_one->next = buff\_two;

p\_two->next = buff\_one;

buff\_two->prev = p\_one; // Предыдущий для второго элемента - предыдущий для первого элемента

buff\_one->prev = p\_two; // Предыдущий для первого элемента - предыдущий для второго элемента

p\_one = buff\_one->next; // Адрес элемента следующего за первым

p\_two = buff\_two->next; // Адрес элемента следующего за вторым

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; // Адрес второго элемента

buff\_one = p\_one; // Адрес первого элемента

HEAD->first = buff\_two;

p\_two->next = buff\_one;

buff\_two->prev = p\_one;

buff\_one->prev = p\_two;

p\_one = buff\_one->next; // Адрес следующего элемента

p\_two = buff\_two->next; // Адрес следующего элемента

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;

}}

// Возвращает 1, если left>right, -1 - если left>right, 0 - в случае равенства

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

int result; //Результат сравнения

switch (type)

{

case 1:

result = strcmp((left->info).name, (right->info).name);

break;

case 2:

result = strcmp((left->info).subject, (right->info).subject);

break;

case 3:

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

result = 1;

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

result = -1;

else

result = 0;

break;

case 4:

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

result = 1;

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

result = -1;

else

result = 0;

break;

case 5:

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

result = 1;

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

result = -1;

else

result = 0;

break;

}

return result;}

// Сортировка списка

void sort(Head \*HEAD){

int i, j, // Параметры цикла для перебора элементов

type; // Тип поля, по которому сортируется список

int decrease; // По убыванию (==1) или же по возрастанию (!=1)

Node \*p=NULL, \*buff=NULL, \*temp = NULL; // Указатель на новый элемент списка, временное хранение указателя списка

do

{

printf("Enter the field number to sort[from 1 to 5]\n");

printf("1 - Name, 2 - Subject, 3 - Price, 4 - Qualification, 5 - Rating\n");

type = get\_int();

} while (type<1 || type>5);

printf("Sort Descending? Enter 1 - Yes, any number - Sort Ascending\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)) // Если buff > p, то поменять их местами

{

swap(HEAD, i, j);

temp = p;

p = buff;

buff = temp->next;

}

else

buff = buff->next;

}

p = p->next;

}}

// Копирование

Node \*copy\_node(Node \*NODE){

int i; // Организация цикла

Node \*p=NULL; // Указатель на новый элемент списка

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

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

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

if((p->info).subject!=NULL && (p->info).name!=NULL)

{

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

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

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

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

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

}

return p;}

// Вывести список в виде таблицы

void print\_tutors(Head \*my\_head){

int i;

Node \*p; // Указатель на элемент списка

printf("|Num| Tutor`s name | Subject | Price per hour |Qualification| Rating |\n");

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

p = my\_head->first;

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

{

printf("|%3d|%22.22s|%16.16s|%16d|%13c|%8.2f|\n", i+1, (p->info).name,

(p->info).subject, (p->info).price, (p->info).qual, (p->info).rating);

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

p = p->next;

}}

// Фильтрация

Head \*selected(Head \*my\_head){

Head \*NEW\_HEAD = NULL; // Указатель на новую "голову" списка

Node \*p = NULL, \*copy\_p = NULL; // Указатель на новый элемент списка и его копию

int i; // Параметр цикла

int maxPrice = 0; // Максимально допустимая цена

float minRating = 0; // Минимально допустимый рейтинг

char \*subject = NULL; // Искомый предемет

NEW\_HEAD = make\_head();

printf("Select subject that you need:\n");

subject = get\_subject();

printf("Select max price per hour(RUB): ");

maxPrice = get\_int();

printf("Select min tutor`s rating: ");

minRating = get\_float();

p = my\_head->first;

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

{

if (((p->info).price <= maxPrice) && ((p->info).rating >= minRating) && (strcmp((p->info).subject, subject)==0)) {

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

p = p->next;

}

return NEW\_HEAD;}

// Очистка для конкретной записи

Node \*clean\_node(Node \*node){

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

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

(node->info).subject = NULL;

(node->info).name = NULL;

free(node);

return NULL;}

// Очистка всего мусора

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

Node \*p, \*temp; // Указатель на элемент списка и временный указатель

int i; // Организация цикла

p = HEAD->first;

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

temp = p;

p = p->next;

temp->next = NULL;

temp->prev = NULL;

temp = clean\_node(temp); }

HEAD->count = 0;

free(HEAD);

return NULL;}

// 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.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 - ");

Q = get\_int();

printf("\n");

return Q;

}

// Справка

void Help(){

system(CLEAR);

printf("\tHELP\n");

printf(" We'll help you chosen a tutor. At first, upload(Enter)\ndatabase using menu item 1. Filter database using menu item 4.\n");

printf(" To get the result of program use menu item 5.\n To finish work use menu item 0.\n");}

# 8. Пример работы программы

## Исходные данные

Таблица 24. Исходные данные

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 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 25. Исходные данные

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | 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 |

## 8.1. Вывод программы

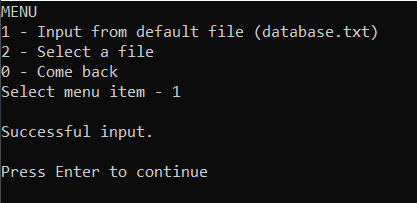


Рисунок 4. Подменю ввода из файла

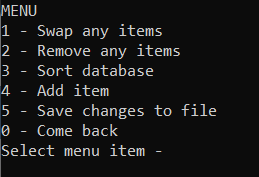


Рисунок 5. Подменю действий со списком

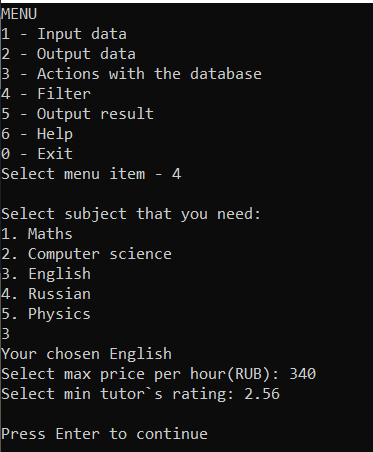


Рисунок 6. Фильтрация списка репетиторов

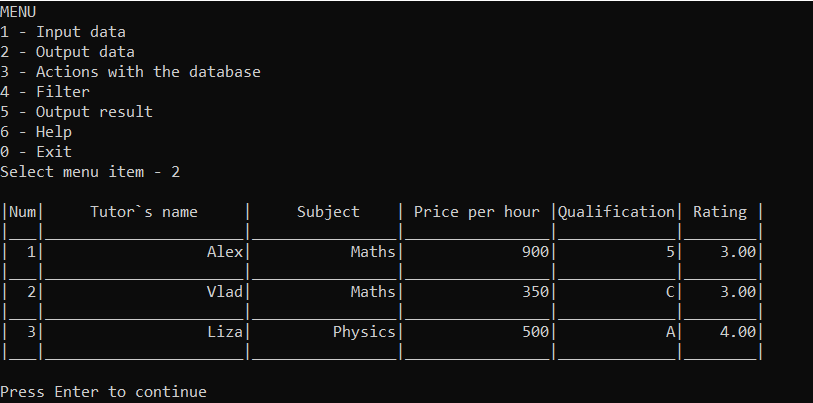


Рисунок 6. Пример вывода списка

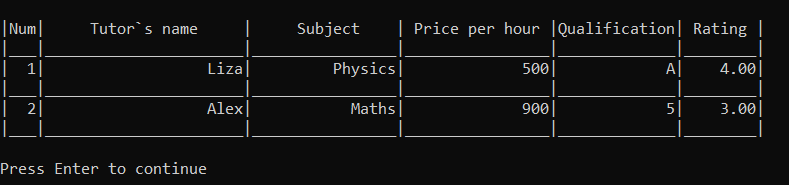


Рисунок 7. Действия со списком согласно таблице 25

## 8.2 Результат работы программы

При выполнении программы получены результаты, совпадающие со значениями, приведёнными в Таблице 24-25. Ошибки не обнаружены. Пример протокола выполнения программы приведены на рисунках 4-7.

# Заключение

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