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

**Отчет по лабораторной работе № 3**

**по дисциплине «Программирование» на**

**тему:** **«Двусвязные списки в языке C/C++»**

|  |  |
| --- | --- |
| Выполнил студент гр. 9308 | Дементьев Д.П. |
|  |  |
| Проверил к.т.н., доцент | Сискович Т.И |

Санкт-Петербург, 2020

Оглавление

**[Введение 3](#_Введение)**

**[1. Задание 3](#_Toc32704)**

**[2. Уточнение задания 3](#_Toc21295)**

**[3. Контрольные примеры 6](#_Toc18046)**

**[4. Краткое описание алгоритма 7](#_Toc20489)**

**[5. Структура вызова функций 8](#_Toc11276)**

**[6. Функции 9](#_Toc11405)**

[6.1 Функция main 9](#_Toc16158)

[6.2 Функция get\_string 1](#_Toc5658)2

[6.3 Функция get\_int 1](#_Toc17485)3

[6.4 Функция get\_float 1](#_Toc17073)4

[6.5 Функция get\_subject 1](#_Toc1338)5

[6.6 Функция add\_item 1](#_Toc14962)6

[6.7 Функция make\_head 1](#_Toc17028)7

[6.8 Функция create\_node 1](#_Toc6285)8

[6.9 Функция add\_first 1](#_Toc23393)9

[6.10 Функция add\_last 20](#_Toc21767)

[6.11 Функция insert 21](#_Toc12003)

[6.12 Функция swap 22](#_Toc25949)

[6.13 Функция remove\_node 23](#_Toc27765)

[6.14 Функция copy\_node 24](#_Toc13194)

[6.15 Функция clean\_node 25](#_Toc30050)

[6.16 Функция clean\_list 2](#_Toc9729)6

[6.17 Функция print\_tutors 2](#_Toc16221)7

[6.18 Функция selected 2](#_Toc2191)8

[6.19 Функция sort 2](#_Toc26713)9

[6.20 Функция compare 3](#_Toc25929)0

[6.21 Функция Menu 3](#_Toc28226)1

**[8. Текст программы 3](#_Toc5615)2**

**[8. Пример работы программы 4](#_Toc20814)9**

[8.1. Вывод программы 4](#_Toc24594)9

[8.2 Результат работы программы 52](#_Toc21134)

**[9. Заключение 53](#_Toc16867)**

# Введение

Целью лабораторной работы является приобретение практических навыков в реализации двусвязных списков на языке программирования C/C++.

# 1. Задание

Написать программу, выполняющую ввод двусвязного списка, содержащих характеристики объектов выбранной предметной области, размер которого заранее неизвестен. Реализовать действия со списком: добавление элемента в начало, конец, на определенную позицию списка. Осуществить контрольный вывод исходной последовательности, сформировать новую последовательность из элементов исходной, поля которой удовлетворяют заданным условиям и вывести полученный результат.

# 2. Уточнение задания

При выполнении задания необходимо учитывать:

1. Вводимый текст не может быть пустым.
2. Максимальная длина вводимой строки – 24 символа.
3. При вводе данных пользователь имеет выбор, куда добавить элемент: в начало, конец или на определенную позицию списка.
4. Если пользователь выбрал вставку элемента на определенную позицию списка, необходимо уточнить, на какую именно, при этом выведя диапазон доступных позиций списка.
5. Предметная область - база репетиторов, в которой есть такие критерии как имя, цена за час, предмет преподавания, квалификация, рейтинг. Исходные данные сортируются по критериям предмета, максимальной цены, минимального рейтинга.

Для данной предметной области были выбраны следующие поля структур:

Для структуры tutor:

Таблица 1. Поля 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:

Таблица 2. Поля структуры tutor

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

Для структуры Head:

Таблица 3. Поля структуры tutor

|  |  |  |
| --- | --- | --- |
| **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:

1. Maths

2. Computer science

3. English

4. Russian

5. Physics

Меню программы должно иметь следующую иерархию:

1 - Ввод исходных данных

2 - Вывод исходных данные

3 - Действия с базой данных

4 - Сортировка исходных данных

5 - Вывод отсортированных данных

6 - Справка

0 - Выход

# 3. Контрольные примеры

Контрольные примеры приведены в таблице 4 и 5.

Таблица 4. 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 5. 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 на экран.

Шаг №2. Выбор пользователем одного из пунктов меню.

Шаг №3. Переход к пункту, выбранным пользователем:

1 - Ввод исходных данных. Переход к шагу 4.

2 - Вывод исходных данных. Переход к шагу 4.

3 - Действия с базой данных. Переход к шагу 4.

4 - Сортировка исходных данных. Переход к шагу 4.

5 - Вывод отсортированных данных. Переход к шагу 4.

6 - Справка. Переход к шагу 4.

0 - Выход. Переход к шагу 4.

Шаг №4. Если пользователь не захотел выйти - переход к шагу 1, иначе конец программы.

Конец.

# 5. Структура вызова функций

Рисунок 1. Структура вызова функций

# struct_main_1 (1)6. Функции

## 6.1 Функция main

**Назначение:**

Является точкой входа в программу.

**Описание переменных:**

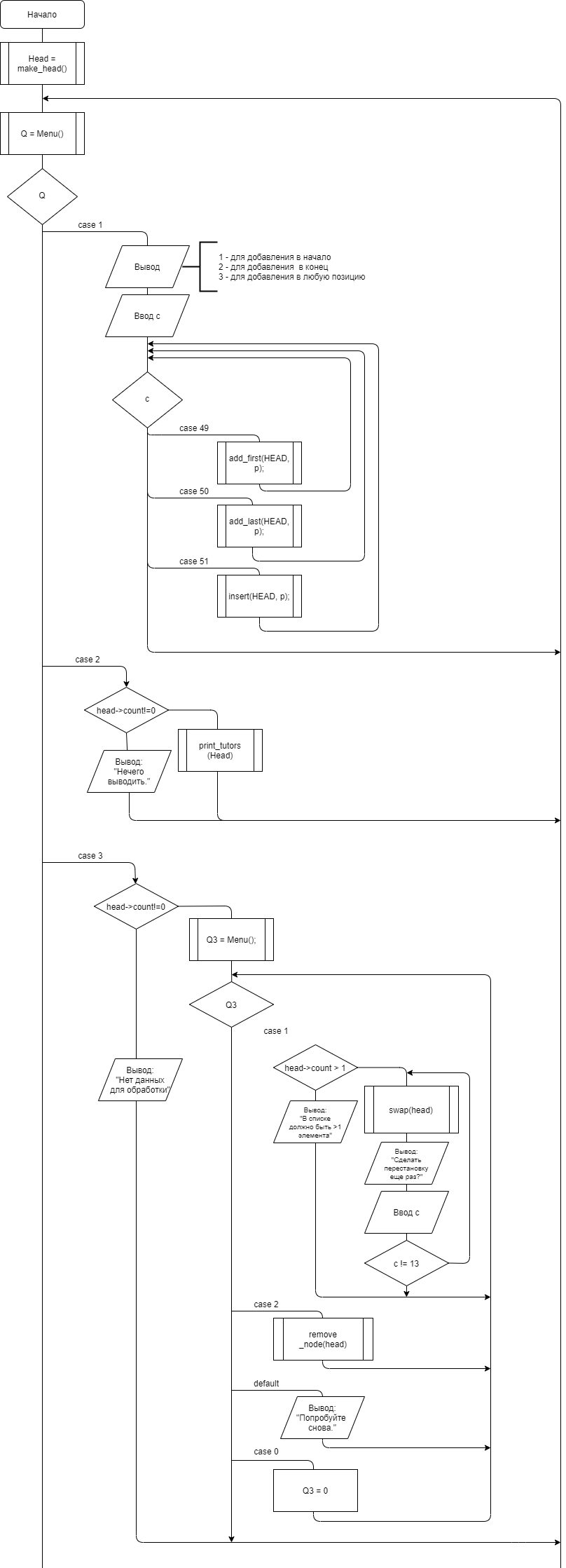
Описание переменных приведено в таблице 6.

Таблица 6. Описание переменных функции main

|  |  |  |
| --- | --- | --- |
| Имя переменной | Тип | Назначение |
| Q3 | int | Номер пункта подменю |
| Q | int | Номер пункта меню |
| c | char | Символ, вводимый с клавиатуры |
| p | Node\* | Проверка выделения памяти для входных данных |
| output | int | Проверка выделения памяти для выходных данных |
| HEAD | Head\* | Указатель на голову списка |
| NEW\_HEAD | Head\* | Указатель на новую голову списка |

**Схема алгоритма:**

Схема алгоритма для функции main представлена на рисунке 2.



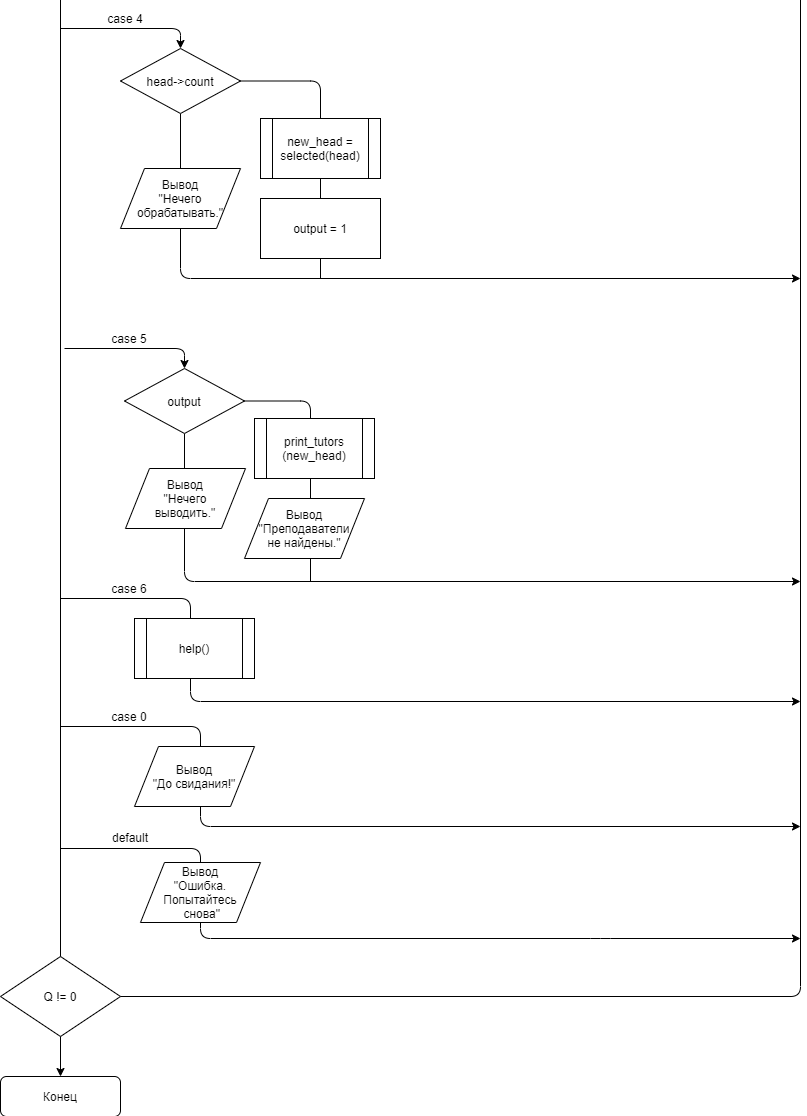


Рисунок 2. Схема алгоритма

## 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 7.

Таблица 7. Описание переменных функции get\_string

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | c | char | Символ строки |
| string | сhar\* | Указатель на первый символ 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 8.

Таблица 8. 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 9.

Таблица 9. 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 10.

Таблица 10. Описание переменных функции get\_subject

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | choice | char\* | Указатель на первый символ строки, выбранного пользователем предмета |
| c | char | Проверка на конец файла |
| q | int | Номер выбранного предмета |
| i | int | Счетчик цикла |

## 6.6 Функция add\_item

**Назначение:**

Функция добавления новой структуры.

**Прототип:**

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

**Пример вызова:**

add\_item(data + ++(\*count) -1);

**Описание переменных:**

Описание переменных приведено в таблице 11.

Таблица 11. Описание переменных функции add\_item

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | list | tutor\* | Указатель на новую структуру |
| Локальная | с | char | Проверка конца строки |

## 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 12.

Таблица 12. Описание переменных функции make\_head

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

## 6.8 Функция create\_node

**Назначение:**

Создание элемента списка.

**Прототип:**

Node \*create\_node()

**Пример вызова:**

p = create\_node();

**Описание переменных:**

Описание переменных приведено в таблице 13.

Таблица 13. Описание переменных функции сreate\_node

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

**6.9 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 14.

Таблица 14. 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 |

**6.10 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 15.

Таблица 15. 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 |

**6.11 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 16.

Таблица 16. 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 |

**6.12 Функция swap**

**Назначение:**

Перестановка двух элементов в списке.

**Прототип:**

void swap(Head \*HEAD);

**Пример вызова:**

swap(HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 17.

Таблица 17. 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 |

**6.13 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 18.

Таблица 18. 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 |
| c | char | Символ, вводимый пользователем с клавиатуры |
| p | Node\* | Указатель на узел |
| buff | Node\* | Буфер |

**6.14 Функция copy\_node**

**Назначение:**

Копирование элемента.

**Прототип:**

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

**Пример вызова:**

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

**Описание переменных:**

Описание переменных приведено в таблице 19.

Таблица 19. 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 |

**6.15 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 20.

Таблица 20. Описание переменных функции clean\_node

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

**6.16 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 21.

Таблица 21. 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 |

## 6.17 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 22.

Таблица 22. 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 |

## 6.18 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 23.

Таблица 23. 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 |

**6.19 Функция sort**

**Назначение:**

Сортировка списка по произвольному полю.

**Прототип:**

void sort(Head \*HEAD);

**Пример вызова:**

sort(HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 24.

Таблица 24. 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 | char | Флаг сортировки по возрастанию/убыванию |
| p | Node\* | Адрес второго элемента |
| buff | Node\* | Буфер |
| temp | Node\* | Временная переменная |
| type | int | Тип поля, по которому сортируется список |

## 6.20 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 25.

Таблица 25. Описание переменных функции compare

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

## 6.21 Функция Menu

**Назначение:**

Функция вывода меню.

**Прототип:**

int Menu();

**Пример вызова:**

Q **=** Menu();

**Описание переменных:**

Описание переменных приведено в таблице 26.

Таблица 26. Описание переменных функции Menu

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | Q | int | Номер пункта меню, выбираемый пользователем |

# 8. Текст программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <malloc.h>

#include <conio.h>

#include <string.h>

#include <windows.h>

#define MAXLEN 24

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

// 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 \*add\_item(tutor \*list); // Заполнение информационных полей у репетитора

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

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

// Интерфейс //

int Menu(); // Вывод меню и выбор его пункта

void Help(); // Справка

int main()

{

Head \*HEAD = NULL, \*NEW\_HEAD = NULL;

Node \*p = NULL;

HEAD = make\_head();

int Q, Q3,output = 0;

int first, second;

char c = 0;

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

{

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

c = getch();

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

switch (c)

{

case 49:

add\_first(HEAD, p);

break;

case 50:

add\_last(HEAD, p);

break;

case 51:

insert(HEAD, p);

break;

case 13:

printf("Successful input.\n");

break;

default:

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

}

} while (c != 13);

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

{

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>second)

puts("The first number must be less that the second.");

} while (first<1 || second>HEAD->count || first>=second);

swap(HEAD, first, second);

puts("Once more swap? Press Enter - No, press any key - Yes");

c = getch();

} while (c != 13);

else

{

Q3=0;

puts("The list must have more than one item.");

}

break;

case 2:

remove\_node(HEAD);

break;

case 3:

sort(HEAD);

break;

case 0:

Q3=0;

break;

default:

puts("Try again!");

break;

}

} while (Q3 != 0);

}

else

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

break;

case 4: //process

if (HEAD->count)

{

if (NEW\_HEAD)

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

NEW\_HEAD = selected(HEAD);

output = 1;

}

else

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

break;

case 5: //output result

if (output)

if (NEW\_HEAD->count) print\_tutors(NEW\_HEAD);

else printf("Tutors not found.\n");

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;

}

system("pause");

} while (Q);

// Очистка всего мусора

if (HEAD->count)

HEAD = clean\_list(HEAD);

if (NEW\_HEAD)

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

return 0;

}

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;

}

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, c;

int q, i;

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

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

do

{

scanf("%d", &q);

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\*));

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

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

return choice;

}

void \*add\_item(tutor \*list) // Добавить элемент в список

{

system("cls");

char c;

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();

do

{

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

list->price = get\_int();

} while (list->price < 0);

printf("Enter qualifications: \n");

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

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

do

{

printf("Enter tutor's rating[From 0 to 5]: \n");

list->rating = get\_float();

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

}

}

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

{

Head \*ph=NULL;

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

if(ph!=NULL)

{

ph->count=0;

ph->first=NULL;

ph->last=NULL;

}

return ph;

}

Node \*create\_node() // Создать заполненную запись

{

Node \*new\_node=NULL;

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

if(new\_node)

{

add\_item(&(new\_node->info));

}

new\_node->prev = NULL;

new\_node->next = NULL;

return new\_node;

}

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++;

}

}

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++;

}

}

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

{

int i, pos;

Node \*p;

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++;

}

}

}

}

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

{

Node \*p;

int i, pos;

char c;

printf("Do you want see list of tutors? Press 1 if you want or press any key otherwise\n");

c = getch();

if (c == 49)

print\_tutors(my\_head);

do

{

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? Press 1 - Yes, any key - No\n");

c = getch();

}

else

c=49;

} while (c==49 && my\_head->count>0);

}

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

{

int i;

Node \*p\_one, \*p\_two;

Node \*buff\_one, \*buff\_two;

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;

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;

}

void sort(Head \*HEAD)

{

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

type; // Тип поля, по которому сортируется список

char decrease; // По убыванию (==13) или же по возрастанию (!=13)

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

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? Press Enter - Yes, any key - Sort Ascending");

decrease = getch();

p = HEAD->first;

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

{

buff = p->next;

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

{

if ((decrease==13) ? (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;

}

}

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

{

int i;

Node \*p=NULL;

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).name);

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

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

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

}

return p;

}

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

{

int i;

Node \*p;

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

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

p = my\_head->first;

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

{

printf("|%22.22s|%16.16s|%19d|%13c|%8.2f|\n", (p->info).name, (p->info).subject, (p->info).price, (p->info).qual, (p->info).rating);

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

p = p->next;

}

}

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

{

Head \*NEW\_HEAD = NULL;

Node \*p = NULL;

int i; // Параметр цикла

int maxPrice = 0;

float minRating = 0;

char \*subject = NULL;

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;

int i;

p = HEAD->first;

HEAD->count = 0;

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

{

temp = p;

p = p->next;

temp->next = NULL;

temp->prev = NULL;

temp = clean\_node(temp);

}

free(HEAD);

return NULL;

}

int Menu(int q)

{

int Q; // Выбор пользователя

system("cls");

puts("MENU");

switch(q)

{

case 0:

puts("1 - Input data");

puts("2 - Output data");

puts("3 - Actions with the database");

puts("4 - Filter");

puts("5 - Output result");

puts("6 - Help");

puts("0 - Exit");

break;

case 3:

puts("1 - Swap any items");

puts("2 - Remove any items");

puts("3 - Sort database");

puts("0 - Come back");

break;

}

printf("Select menu item - ");

scanf("%d", &Q);

printf("\n");

fflush(stdin);

return Q;

}

void Help()

{

system("cls");

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 27. 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 28. 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 3. Пример работы программы

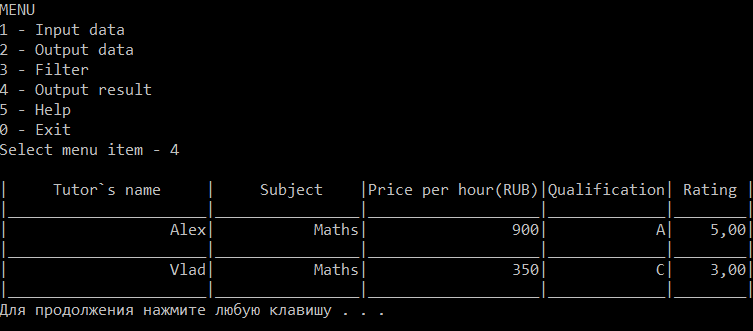


Рисунок 4. Пример работы программы

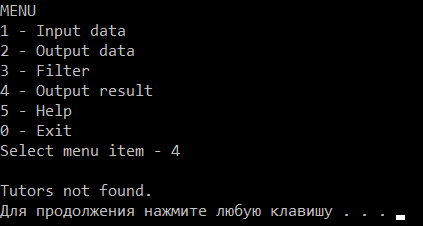


Рисунок 5. Пример работы программы

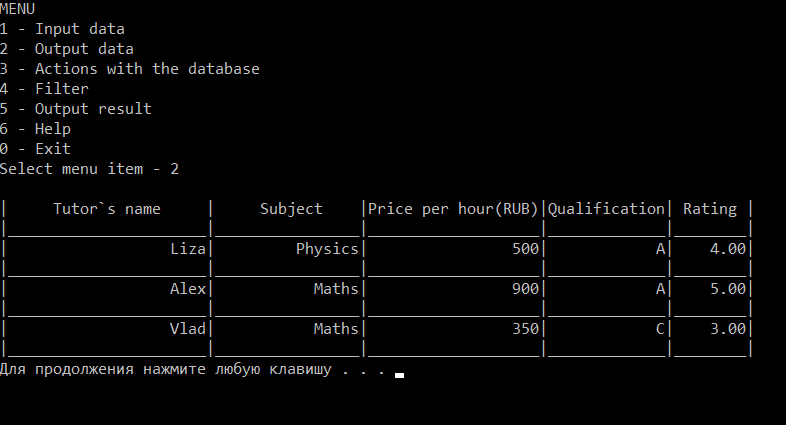
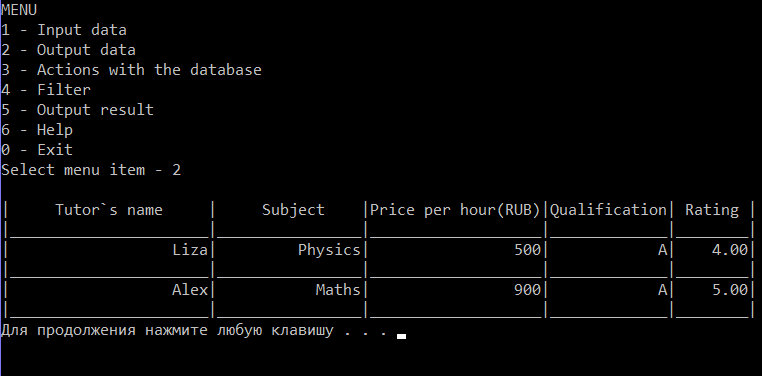


Рисунок 6. Пример работы программы



## 8.2 Результат работы программы

При выполнении программы получены результаты, совпадающие со значениями, приведенными в Таблице 27-28. Ошибки не обнаружены. Пример протокола выполнения программы приведены на рисунках 3-6.

# 9. Заключение

При выполнении лабораторной работы получены практические навыки в реализации двусвязного списка на языке программирования C/C++.