МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе № 8

по дисциплине «Программирование»

ТЕМА: «ЛИНЕЙНЫЕ ОДНОСВЯЗНЫЕ СПИСКИ»

Студент гр. 3311 _____ Шарпинский Д. А.

Преподаватель _____ Хахаев И. А.

Санкт-Петербург

2024

Цель работы.

Научиться работать с линейными односвязными списками в языке С.

Задание (вариант 14)

С использованием структуры, созданной при выполнении лабораторной работы №7 (по выбранной предметной области), создать односвязный линейный список и выполнить задание в соответствии с вариантом.

Разработать подалгоритм и написать функцию, удаляющую в односвязном списке элемент перед элементом с указанным номером. Если указан номер первого элемента, вывести сообщение о невозможности удаления.

ПРЕДМЕТНАЯ ОБЛАСТЬ:

- **Пользователи социальной сети**

Постановка задачи и описание решения

Целью написания программы является демонстрация работы с односвязным списком пользователей, включая основные операции, такие как чтение из файла, добавление, удаление, сортировка и фильтрация данных. Жизненный цикл программы начинается с инициализации списка пользователей путём чтения данных из файла CSV.

Структура User, используемая для хранения информации о пользователях, представлена следующим образом:

Название поля	Тип	Описание
id	int	Уникальный идентификатор пользователя
fullName	char*	Полное имя пользователя
age	int	Возраст пользователя
profession	char*	Профессия пользователя
friendsRating	float	Рейтинг среди друзей
publicRating	float	Общественный рейтинг

friendsCount	int	Количество друзей
friendsId	int*	Массив идентификаторов друзей
next	User*	Указатель на следующего пользователя

А также важной частью программы является структура Head, которая служит для управления списком пользователей:

Название поля	Тип	Описание
last_id	int	Идентификатор последнего пользователя
isFriendsSorted	int	Флаг сортировки списка по количеству друзей
first	User*	Указатель на первого пользователя в списке
last	User*	Указатель на последнего пользователя в списке

После инициализации данных пользователь встречается с меню, предлагающим различные опции для управления списком:

1. вывод всего списка
2. сортировка по количеству друзей
3. сортировка по идентификатору
4. добавление нового пользователя
5. фильтрация по имени или профессии
6. удаление пользователя перед указанным id
7. очистка списка
8. выход из программы

Для сортировки используется алгоритм пузырьковой сортировки, адаптированный для работы со связными списками, что позволяет упорядочивать элементы списка согласно заданным критериям.

Одним из ключевых аспектов программы является управление памятью. Благодаря использованию функций для динамического выделения и освобождения памяти, в программе отсутствуют утечки памяти, что было

проверено с использованием инструмента Valgrind на операционной системе Linux.

Описание переменных

Функция main()

№	Имя переменной	Тип	Назначение
1	head	Head*	Указатель на голову списка, хранит основные данные о списке
2	user	User*	Вспомогательный указатель для работы с пользователями
3	slen	int	Длина строки, используется при чтении данных из файла
4	i	int	Индекс в цикле, общее использование в различных циклах for
5	n	int	Количество строк (записей) в файле CSV
6	count	int	Счётчик успешно добавленных пользователей
7	option	int	Переменная для хранения выбранного пользователем варианта действия в меню
8	temp	char[]	Временная строка для чтения данных из файла
9	splitArray	char**	Массив строк, полученный в результате разбиения строки из файла
10	file	FILE*	Указатель на файл, открытый для чтения данных

Функция makeNode()

№	Имя переменной	Тип	Назначение
1	str	char**	Массив строк с данными для нового пользователя

Функция addNode()

№	Имя переменной	Тип	Назначение
1	my_head	Head*	Указатель на голову списка
2	new_node	User*	Указатель на добавляемого пользователя

Функция selectId()

№	Имя переменной	Тип	Назначение
1	my_head	Head*	Указатель на голову списка
2	id	int	Идентификатор пользователя, которого требуется удалить

Функция deleteNode()

№	Имя переменной	Тип	Назначение
1	my_head	Head*	Указатель на голову списка
2	current_node	User*	Указатель на удаляемого пользователя

Функция deleteById()

№	Имя переменной	Тип	Назначение
1	head	Head*	Указатель на голову списка

Функция addUser()

№	Имя переменной	Тип	Назначение
1	head	Head*	Указатель на голову списка

Функция freeStruct()

№	Имя переменной	Тип	Назначение
1	user	User*	Указатель структуру, для которой требуется провести операцию очистки памяти

Функция freeList()

№	Имя переменной	Тип	Назначение
1	my_head	Head*	Указатель на голову списка

Функция bubbleSortByField()

№	Имя переменной	Тип	Назначение
1	my_head	Head*	Указатель на голову списка
2	desc	char*	Строка с названием поля, по которому производится сортировка

Функция swapNodes()

№	Имя переменной	Тип	Назначение
1	prevNode	User**	Указатель на предыдущий элемент в списке
2	a	User*	Указатель на текущий элемент для обмена
3	b	User*	Указатель на следующий элемент для обмена

Функция startsWithIgnoreCase()

№	Имя переменной	Тип	Назначение
1	str	char*	Строка, в которой ищется подстрока
2	prefix	char*	Подстрока, по которой производится поиск

Функция filterList()

№	Имя переменной	Тип	Назначение
1	head	Head*	Указатель на голову списка

Функция clearList()

№	Имя переменной	Тип	Назначение
1	head	Head*	Указатель на голову списка

Функция simpleSplit()

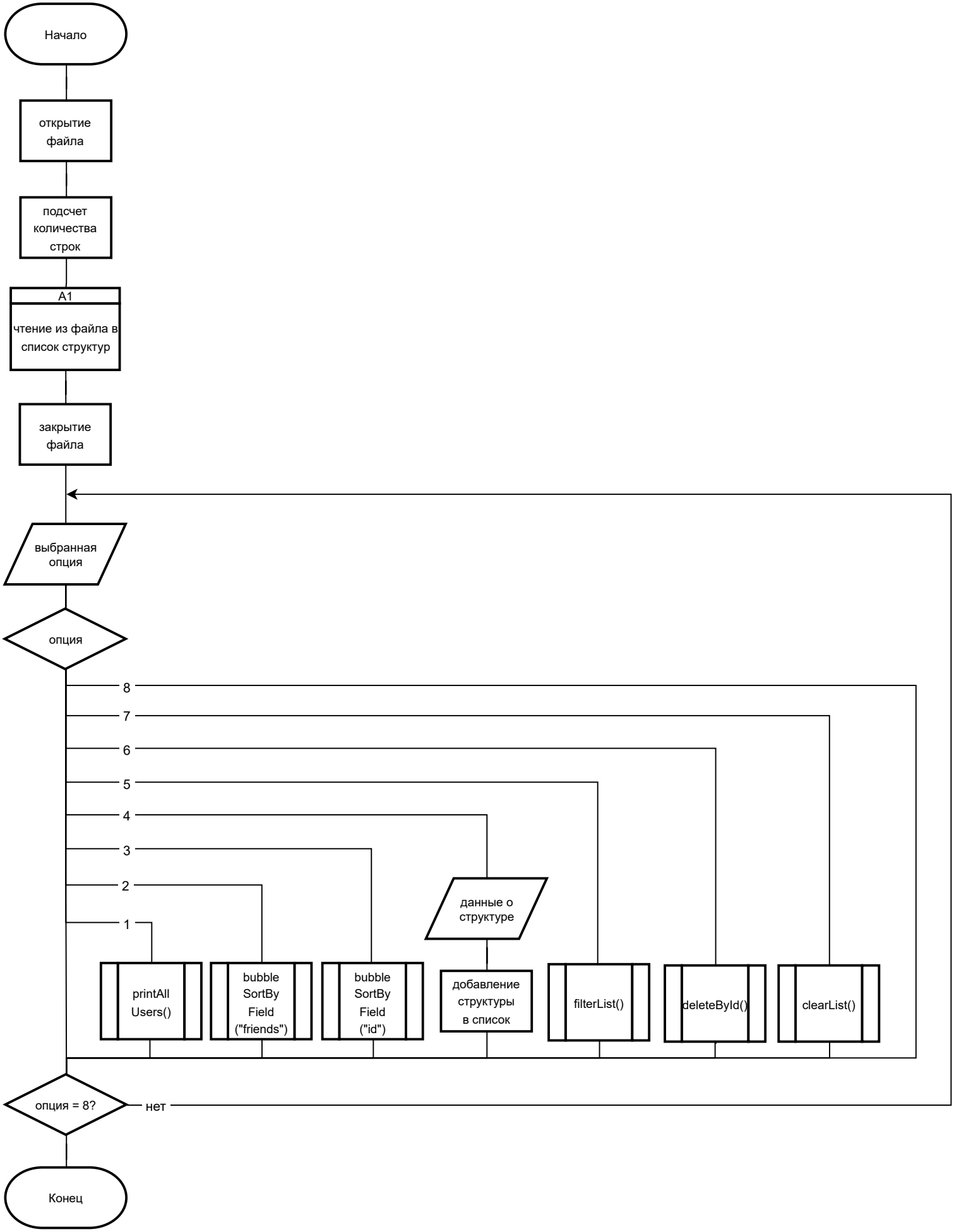
№	Имя переменной	Тип	Назначение
1	str	char*	Строка для разбиения
2	length	int	Длина строки

Функция simpleSplitInt()

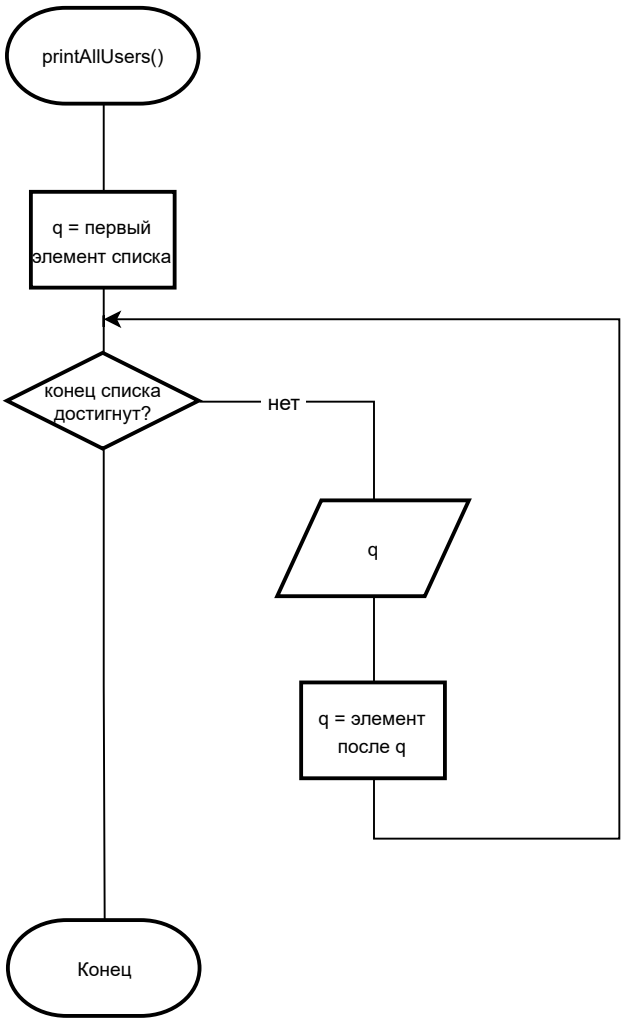
№	Имя переменной	Тип	Назначение
1	user	User*	Указатель структуру, для которой требуется провести операцию очистки памяти
2	str	char*	Строка для разбиения
3	isManual	int	Флаг, указывающий, требуется ли сопоставление количества друзей, введенных в строке, и количества, указанного пользователем
4	idList	int[]	Массив всех идентификаторов пользователей
5	usersCount	int	Количество всех пользователей

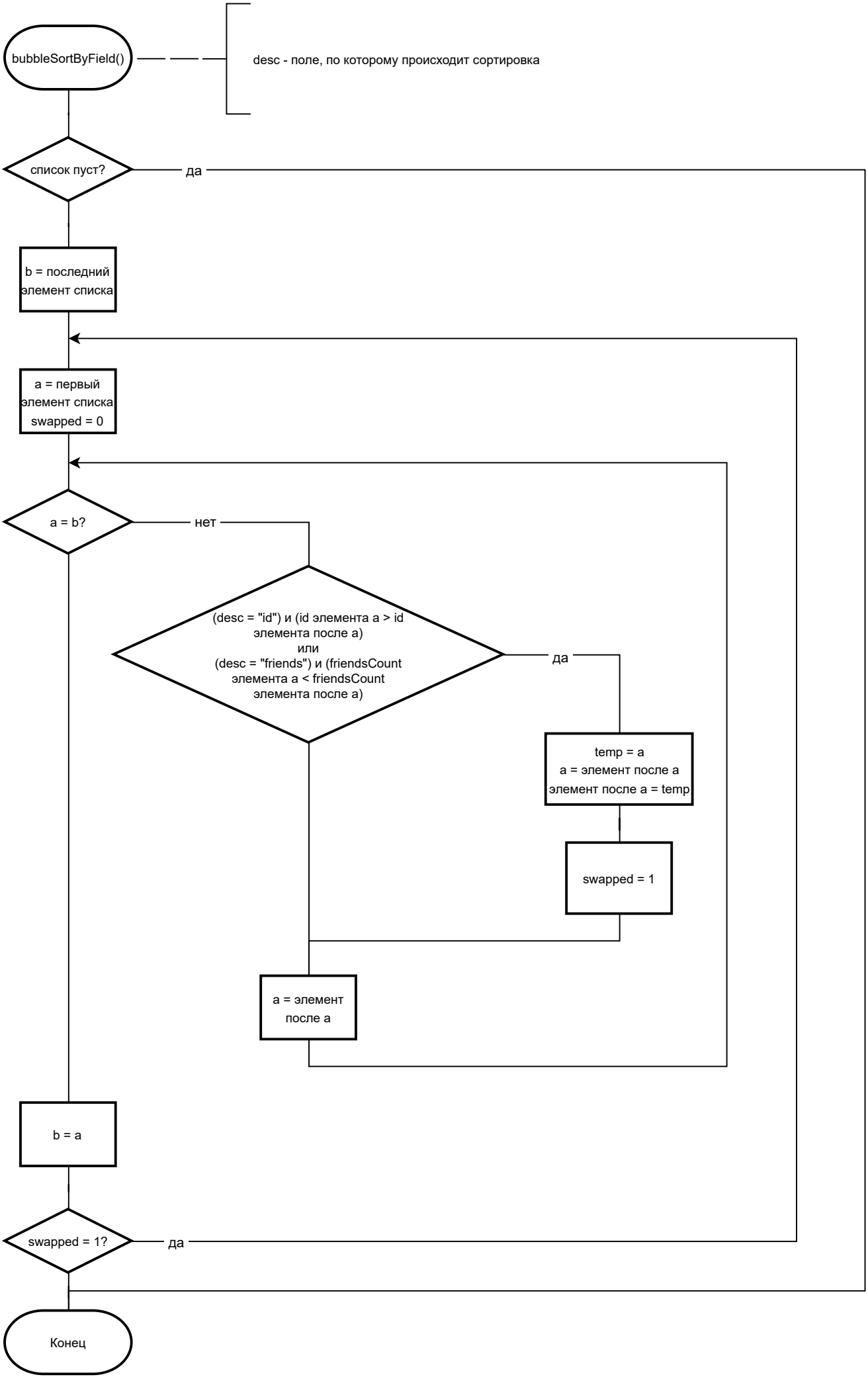
Функция trim()

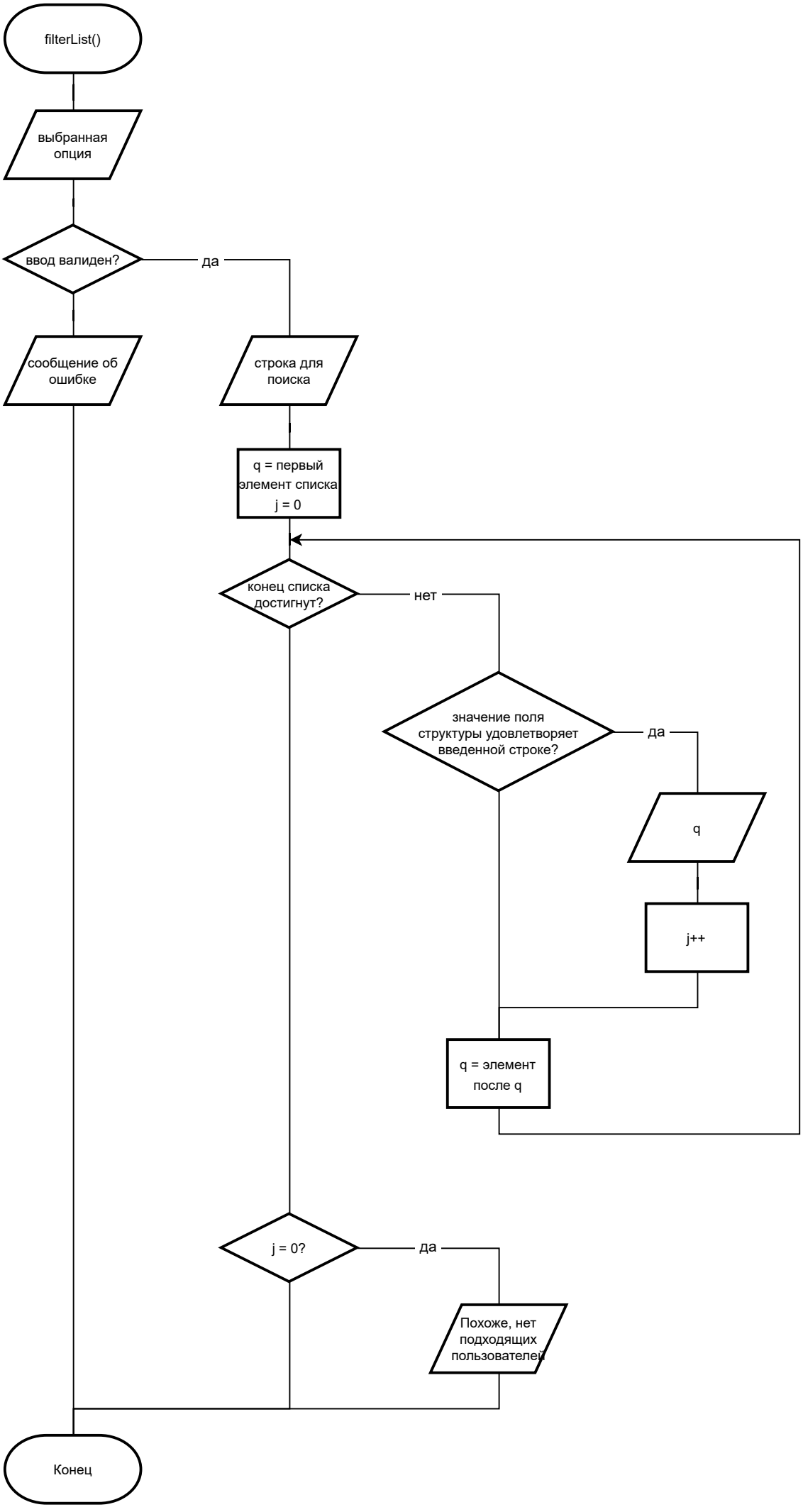
№	Имя переменной	Тип	Назначение
1	str	char*	Строка для обрезки

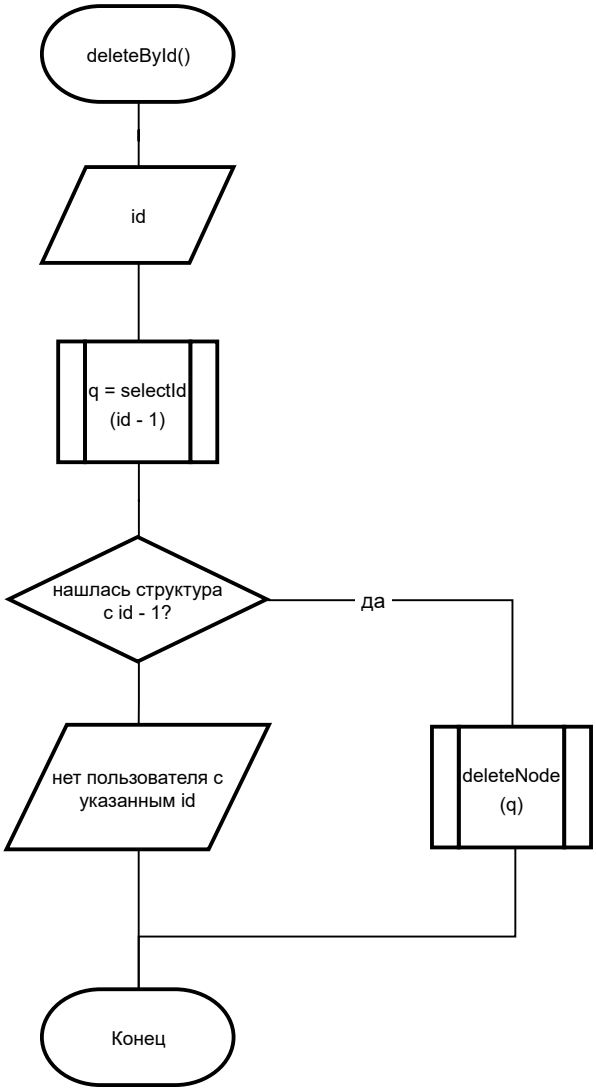

Начало
открытие файла
подсчет количества строк
A1
чтение из файла в список структур
закрытие файла
выбранная опция
опция
8
7
6
5
4
3
2
1
данные о структуре
printAll Users()
bubble SortBy Field ("friends")
bubble SortBy Field ("id")
добавление структуры в список
filterList()
deleteById()
clearList()
опция = 8?
нет
Конец

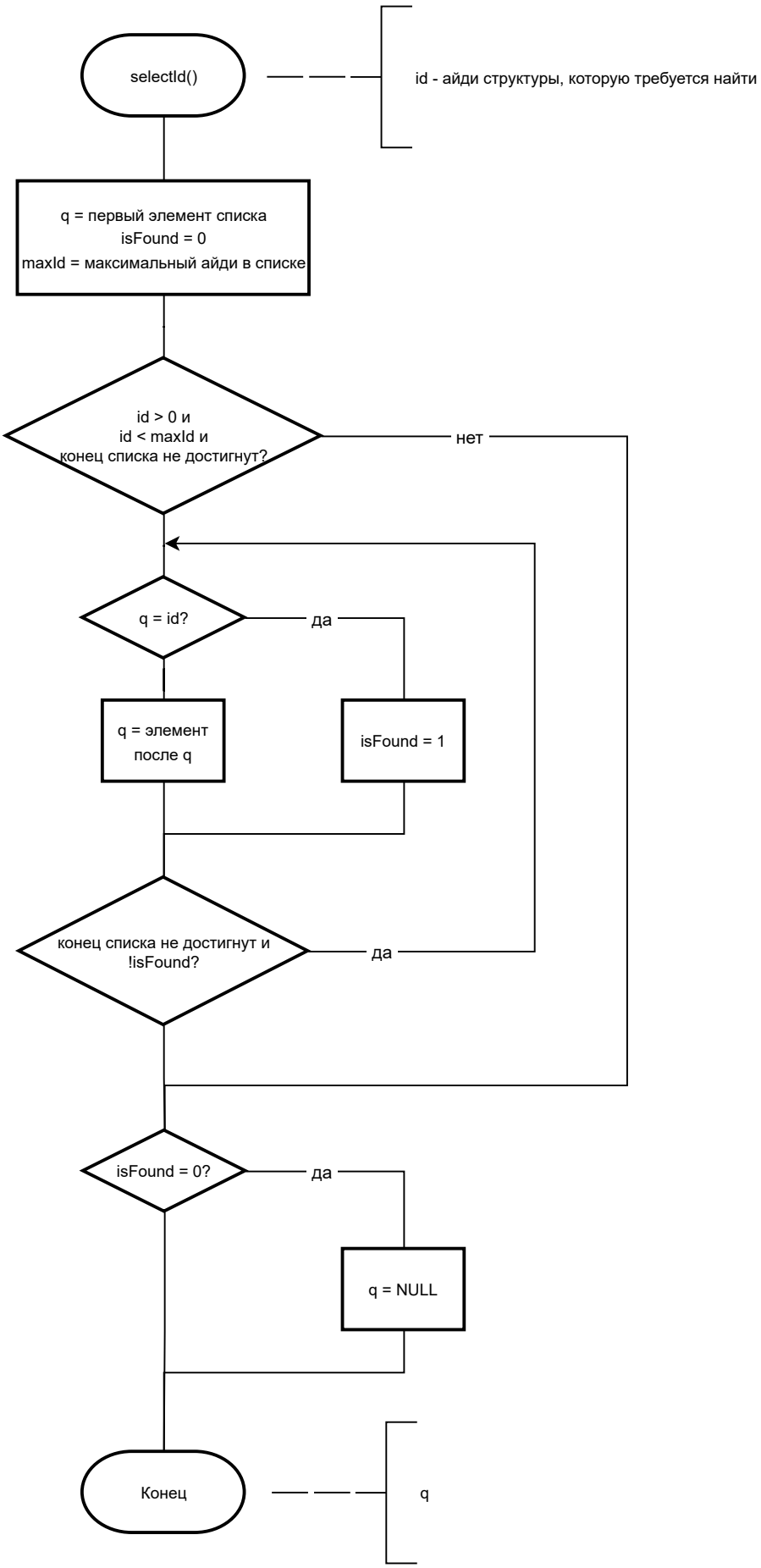

Начало A1
i = 0
i < n?
да
строка из файла
разбиение строки по разделителям
заполнение i-ой структуры
добавление структры в список
i++
Конец A1

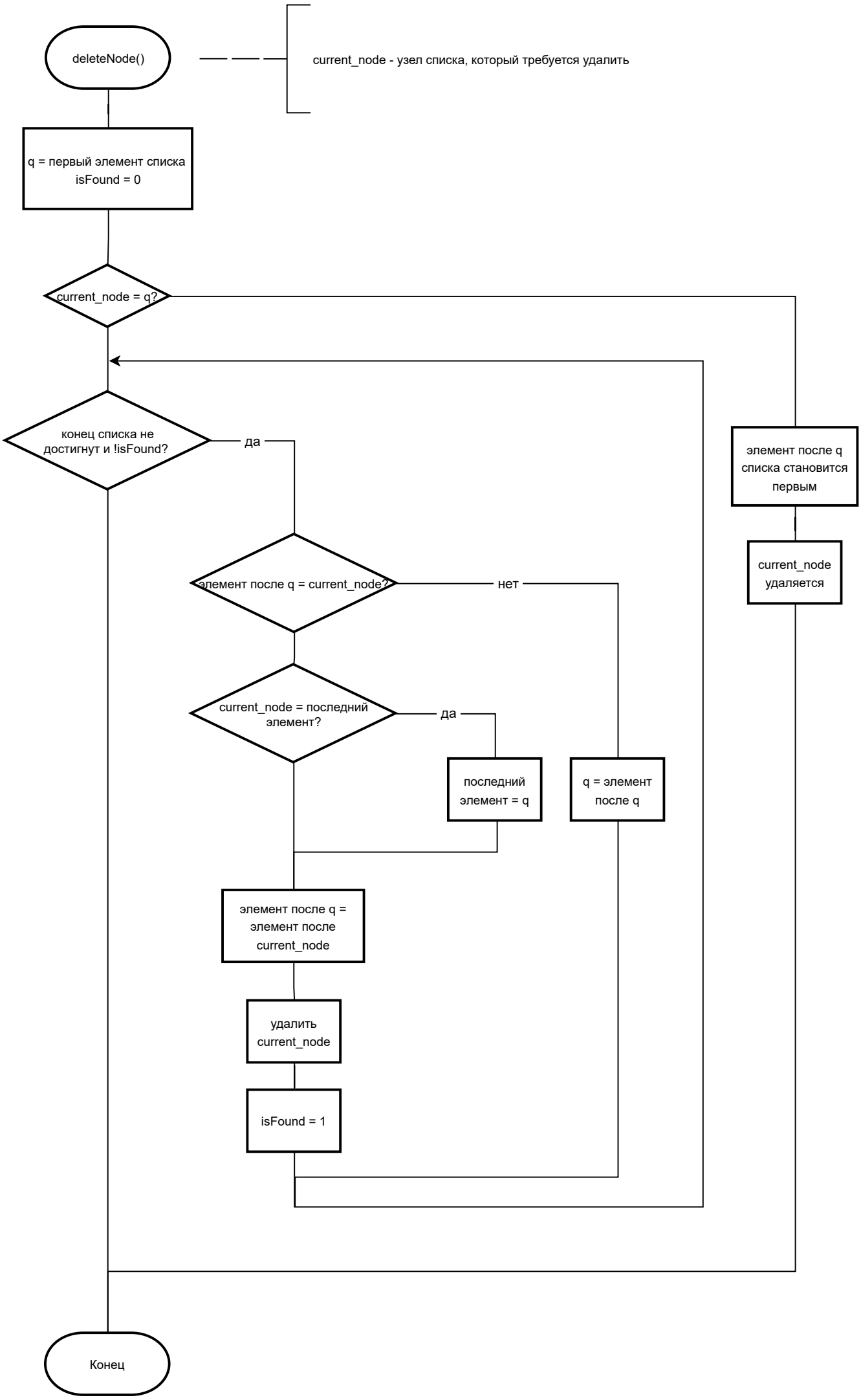

printAllUsers()
q = первый элемент списка
конец списка достигнут?
нет
q
q = элемент после q
Конец

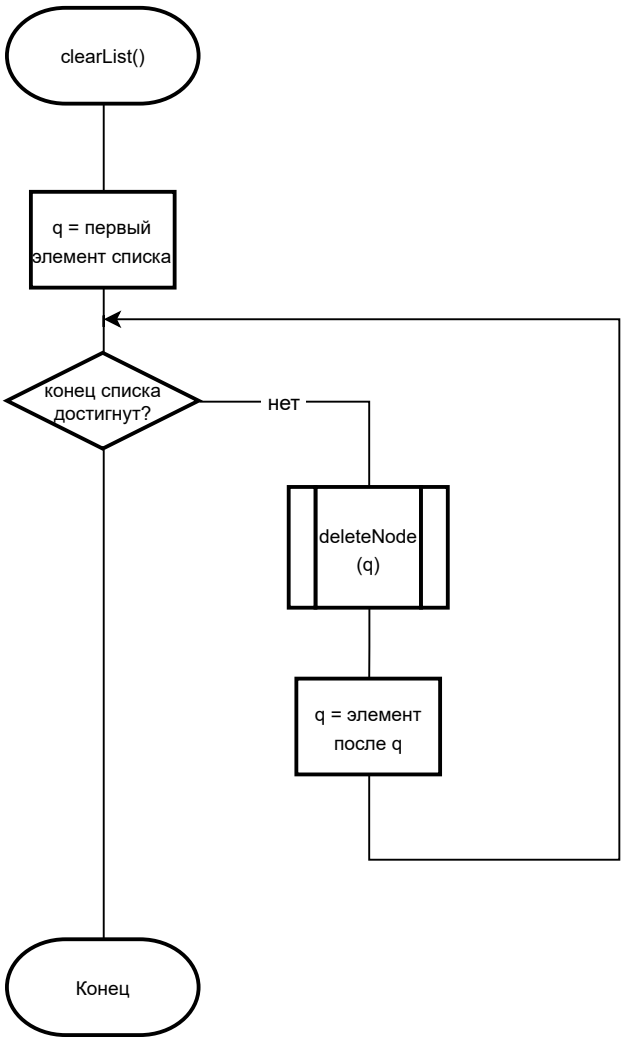

bubbleSortByField()
desc - поле, по которому происходит сортировка
список пуст?
да
b = последний элемент списка
a = первый элемент списка
swapped = 0
a = b?
нет
(desc = "id") и (id элемента a > id элемента после a)
или
(desc = "friends") и (friendsCount элемента a < friendsCount элемента после a)
да
temp = a
a = элемент после a
элемент после a = temp
swapped = 1
a = элемент после a
b = a
swapped = 1?
да
Конец


filterList()
выбранная опция
ввод валиден?
да
сообщение об ошибке
строка для поиска
q = первый элемент списка
j = 0
конец списка достигнут?
нет
значение поля структуры удовлетворяет введенной строке?
да
q
j++
q = элемент после q
j = 0?
да
Похоже, нет подходящих пользователей
Конец


deleteById()
id
q = selectId
(id - 1)
нашлась структура
с id - 1?
да
нет пользователя с
указанным id
deleteNode
(q)
Конец


selectId()
id - айди структуры, которую требуется найти
q = первый элемент списка
isFound = 0
maxId = максимальный айди в списке
id > 0 и
id < maxId и
конец списка не достигнут?
нет
q = id?
да
q = элемент
после q
isFound = 1
конец списка не достигнут и
!isFound?
да
isFound = 0?
да
q = NULL
Конец
q


deleteNode()
current_node - узел списка, который требуется удалить
q = первый элемент списка
isFound = 0
current_node = q?
конец списка не достигнут и !isFound?
да
элемент после q списка становится первым
current_node удаляется
элемент после q = current_node?
нет
current_node = последний элемент?
да
последний элемент = q
q = элемент после q
элемент после q = элемент после current_node
удалить current_node
isFound = 1
Конец


clearList()
q = первый элемент списка
конец списка достигнут?
нет
deleteNode (q)
q = элемент после q
Конец

Исходный текст программы

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

#define MAXLEN 256

typedef struct userStruct {
    int id;
    char* fullName;
    int age;
    char* profession;
    float friendsRating;
    float publicRating;
    int friendsCount;
    int friendsId[MAXLEN];
    struct userStruct* next;
} User;

typedef struct LHead {
    int last_id;
    int isFriendsSorted;
    struct userStruct* first;
    struct userStruct* last;
} Head;

/** for work with linked list */
Head* makeHead();
User* makeNode(char** str);
void addNode(Head* my_head, User* new_node);
User* selectId(Head* my_head, int id);
void deleteNode(Head* my_head, User* current_node);
void deleteById(Head* head);
void addUser(Head* head);
void freeStruct(User* user);
void freeList(Head* my_head);

/** utils */
void bubbleSortByField(Head *my_head, char* desc);
```

```
void swapNodes(User** prevNode, User* a, User* b);
int startsWithIgnoreCase(const char* str, const char* prefix);
void filterList(Head* head);
void clearList(Head* head);
char** simpleSplit(char* str, int length);
void simpleSplitInt(User* user, const char* str, int isManual, int idList[], int usersCount);
void trim(char* str);

/** output */
void printHeader();
void printUser(User* user);
void printAllUsers(Head* my_head);
void pressEnterToContinue();
void clearConsole();

int main() {
    Head* head = NULL;
    User* user = NULL;
    int slen, i, n, count, option;
    char temp[MAXLEN];
    char** splitArray;
    FILE* file;

    head = makeHead();
    n = 0;
    count = 0;
    file = fopen("index.csv", "r");
    if (file != NULL) {
        while ((fgets(temp, MAXLEN, file)) != NULL) n++;
        rewind(file);

        for (i = 0, count = 0; i < n; i++, count++) {
            fgets(temp, MAXLEN, file);
            slen = strlen(temp);
            temp[slen - 1] = '\0';

            splitArray = simpleSplit(temp, slen);
            if (splitArray != NULL) {
                user = makeNode(splitArray);
                if (user != NULL) {
```

```
                addNode(head, user);
            } else {
                puts("Structure not allocated!");
                i = n;
            }
        } else {
            puts("Error data reading!");
            i = n;
        }
    }
    fclose(file);
} else {
    perror("Failed to open file");
    printf("\n");
}

if (count == n) {
    bubbleSortByField(head, "id");
    clearConsole();
    do {
        printf("Choose option and press ENTER:\n"
               "1. Print all users\n"
               "2. Sort users by friends count\n"
               "3. Sort users by id\n"
               "4. Add new user\n"
               "5. Filter users by name or profession\n"
               "6. Delete user before specified id\n"
               "7. Clear list\n"
               "8. Exit\n\n");
        printf("Option: ");
        scanf("%d", &option);
        getchar();
        switch (option) {
        case 1:
            clearConsole();
            printAllUsers(head);
            pressEnterToContinue();
            break;
        case 2:
            clearConsole();
```

```
    bubbleSortByField(head, "friends");
    printf("Sorted!\n");
    head->isFriendsSorted = 1;
    pressEnterToContinue();
    break;
case 3:
    clearConsole();
    bubbleSortByField(head, "id");
    printf("Sorted!\n");
    head->isFriendsSorted = 0;
    pressEnterToContinue();
    break;
case 4:
    clearConsole();
    addUser(head);
    pressEnterToContinue();
    break;
case 5:
    clearConsole();
    filterList(head);
    pressEnterToContinue();
    break;
case 6:
    clearConsole();
    if (head->first != NULL) {
        deleteById(head);
    } else {
        printf("The list is cleared and there is no users to remove!\n"
               "You can add new users with command 4 in menu!\n");
    }
    pressEnterToContinue();
    break;
case 7:
    clearConsole();
    if (head->first != NULL) {
        clearList(head);
        printf("The list is now cleared!\n");
    } else {
        printf("The list is already empty!\n");
    }
```

```
                pressEnterToContinue();
            case 8:
                clearConsole();
                break;
            default:
                clearConsole();
                break;
            }
        } while (option != 8);
    } else {
        puts("An error occurred while running the program!");
    }

    freeList(head);

    return 0;
}









Head* makeHead() {
    Head* ph = NULL;
    ph = (Head*)malloc(sizeof(Head));
    if (ph != NULL) {
        ph->last_id = 0;
        ph->first = NULL;
        ph->last = NULL;
        ph->isFriendsSorted = 0;
    }
    return ph;
}

User* makeNode(char** str) {
```

```
    int mock[] = {0};
    User *user = NULL;
    user = (User*)malloc(sizeof(User));

    if (user != NULL) {
        user->id = atoi(str[0]);
        free(str[0]);
        user->fullName = str[1];
        user->age = atoi(str[2]);
        free(str[2]);
        user->profession = str[3];
        user->friendsRating = atof(str[4]);
        free(str[4]);
        user->publicRating = atof(str[5]);
        free(str[5]);
        user->friendsCount = atoi(str[6]);
        free(str[6]);

        simpleSplitInt(user, str[7], 0, mock, 0);
        free(str[7]);

        free(str);

        user->next = NULL;
    }

    return user;
}

void addNode(Head* my_head, User* new_node) {
    if (my_head->first == NULL) {
        my_head->last_id = 1;
        my_head->first = new_node;
        my_head->last = new_node;
    } else {
        my_head->last_id++;
        new_node->id = my_head->last_id;
        new_node->next = my_head->first;
        my_head->first = new_node;
    }
```

```
}

User *selectId(Head *my_head, int id) {
    User *q = NULL;
    int maxId, isFound;

    isFound = 0;
    q = my_head->first;
    maxId = my_head->last_id;

    if ((id > 0) && (id <= maxId) && q != NULL) {
        do {
            if (q->id == id) {
                isFound = 1;
            } else {
                q = q->next;
            }
        } while (q != NULL && !isFound);
    }

    if (!isFound) {
        q = NULL;
    }

    return q;
}

void deleteNode(Head *my_head, User *current_node) {
    User *q;
    int isFound = 0;

    q = my_head->first;

    if (current_node == q) {
        my_head->first = current_node->next;
        current_node->next = NULL;
        freeStruct(current_node);
    } else {
        while (q != NULL && !isFound) {
            if (q->next == current_node) {
```

```
                if (current_node == my_head->last) {
                    my_head->last = q;
                }
                q->next = current_node->next;
                current_node->next = NULL;
                freeStruct(current_node);
                isFound = 1;
            } else {
                q = q->next;
            }
        }
    }

    if (my_head->first != NULL) {
        my_head->last_id = my_head->last->id;
    } else {
        my_head->last_id = 0;
    }
}

void deleteById(Head* head) {
    User* q;
    int id, i, idFlag;

    bubbleSortByField(head, "id");
    q = head->first;

    printf("Enter the users id to remove the user BEFORE it\n"
           "(enter 0 if you don't want to do it)\n\n"
           "Here is list of ids:\n[");
    while (q != NULL) {
        printf("%d", q->id);
        if (q->next != NULL) {
            printf(", ");
        }
        q = q->next;
    }
    printf("]\n\n");
    printf("Example:\n"
           "id: 2 -> user with id 1 will be removed\n\n"
```

```
            "id: ");
    scanf("%d", &id);
    getchar();
    if (id != 0) {
        q = selectId(head, id - 1);
        if (q != NULL) {
            deleteNode(head, q);
            printf("\nSuccess: user with id %d has been removed!\n", id - 1);
            q = head->first;
            while (q != NULL) {
                idFlag = 0;
                for (i = 0; i < q->friendsCount; i++) {
                    if (q->friendsId[i] == (id - 1)) {
                        idFlag = 1;
                        q->friendsCount--;
                        q->friendsId[q->friendsCount] = 0;
                    }
                    if (idFlag) {
                        q->friendsId[i] = q->friendsId[i + 1];
                    }
                }
                q = q->next;
            }
        } else {
            printf("Failed: there is no user with id %d.\n", id - 1);
        }
    }
    if (head->isFriendsSorted) {
        bubbleSortByField(head, "friends");
    }
}

void addUser(Head *head) {
    User *newUser, *q;
    char tempStr[MAXLEN];
    int idList[MAXLEN] = {0};
    int tempAge, i;
    int usersCount = 0;
    float tempFriendsRating, tempPublicRating;
    int tempFriendsCount;
```

```
newUser = malloc(sizeof(User));
if (newUser == NULL) {
    perror("Memory allocation failed");
} else {
    newUser->next = NULL;

    newUser->id = head->last_id + 1;

    printf("Enter full name: ");
    newUser->fullName = malloc(MAXLEN * sizeof(char));
    if (newUser->fullName == NULL || fgets(newUser->fullName, MAXLEN, stdin) == NULL) {
        perror("Failed to read full name or allocate memory");
        free(newUser);
    } else {
        trim(newUser->fullName);
        printf("\nEnter age: ");
        scanf("%d", &tempAge);
        getchar();
        newUser->age = tempAge > 0 && tempAge < 256 ? tempAge : 0;
        if (tempAge >= 256) puts("No way human live more than 256 years -> setting 0 automatically");

        printf("\nEnter profession: ");
        newUser->profession = malloc(MAXLEN * sizeof(char));
        if (newUser->profession == NULL || fgets(newUser->profession, MAXLEN, stdin) == NULL) {
            perror("Failed to read image filename or allocate memory");
            free(newUser->fullName);
            free(newUser);
        } else {
            trim(newUser->profession);
            printf("\nEnter friends rating (float number less than 5): ");
            if (scanf("%f", &tempFriendsRating) == 1 && tempFriendsRating <= 5) {
                newUser->friendsRating = tempFriendsRating;
            } else {
                puts("Invalid friends rating -> setting 0 automatically");
                newUser->friendsRating = 0;
            }
            getchar();

            printf("\nEnter public rating (float number less than 5): ");
```

```
                if (scanf("%f", &tempPublicRating) == 1 && tempPublicRating <= 5) {
                    newUser->publicRating = tempPublicRating;
                } else {
                    puts("Invalid public rating -> setting 0 automatically");
                    newUser->publicRating = 0;
                }
                getchar();

                if (head->first != NULL) {
                    usersCount = 0;
                    bubbleSortByField(head, "id");
                    q = head->first;
                    while (q != NULL) {
                        idList[usersCount++] = q->id;
                        q = q->next;
                    }
                    printf("\nEnter friends count (less than %d): ", usersCount);

                    if (scanf("%d", &tempFriendsCount) == 1 && tempFriendsCount <= MAXLEN &&
tempFriendsCount >= 0 && tempFriendsCount <= usersCount) {
                        newUser->friendsCount = tempFriendsCount;
                    } else {
                        puts("Invalid friends count -> setting 0 automatically");
                        tempFriendsCount = 0;
                        newUser->friendsCount = 0;
                    }
                    getchar();
                } else {
                    newUser->friendsCount = 0;
                    for (i = 0; i < MAXLEN; i++) {
                        newUser->friendsId[i] = 0;
                    }
                }

                if (tempFriendsCount > 0) {
                    printf("\nEnter friends IDs (example: 1,2,3,4)\n");
                    printf("Available IDs: [%d", idList[0]);
                    for (i = 1; i < usersCount; i++) {
                        printf(",%d", idList[i]);
                    }
                    printf("]\n\n");
```

```
                    printf("IDs: ");
                    scanf("%s", tempStr);
                    getchar();
                    simpleSplitInt(newUser, tempStr, 1, idList, usersCount);
                }

                addNode(head, newUser);
                if (head->isFriendsSorted) {
                    bubbleSortByField(head, "friends");
                } else {
                    bubbleSortByField(head, "id");
                }

                printf("\nNew user successfully added!\n");

            }
        }
    }
}

void freeStruct(User *user) {
    if (user != NULL) {
        free(user->fullName);
        user->fullName = NULL;

        free(user->profession);
        user->profession = NULL;

        free(user);
    }
}

void freeList(Head* my_head) {
    User *q, *q1;
    /* there are two pointers here because we need to remember
    the next value of the structure we are going to free */
    q = my_head->first;
    while (q != NULL) {
        q1 = q->next;
        freeStruct(q);
```

```
        q = q1;
    }
    free(my_head);
}
```

```
/**
 * Sorts a linked list of users either by their ID or by their friend count in ascending order.
 * This function implements a bubble sort algorithm that iterates through the list, comparing
 * adjacent nodes based on the specified field (`desc`). If `desc` is "id", nodes are sorted
 * by their ID in ascending order else nodes are sorted by their friendsCount in descending order.
 *
 * The use of a double pointer (**ptr1) for node swapping is crucial here. It allows directly
 * modifying the pointer to the current node in the list (either the `first` field of the head
 * or the `next` field of a node). This direct manipulation eliminates the need for a separate
 * case to update the head of the list when the first two nodes are swapped and simplifies
 * swapping nodes in general by adjusting the pointer that points to the current node rather
 * than the node itself.
 *
 * @param my_head A pointer to the head of the list, which contains pointers to the first
 *                and last nodes of the list.
 * @param desc A string that specifies the field by which the list should be sorted:
 *             "id" for sorting by ID in ascending order, else for sorting by
 *             friendsCount in descending order.
 */
void bubbleSortByField(Head *my_head, char* desc) {
    int swapped;
    User **ptr1;
    User *lptr = NULL;

    if (my_head->first != NULL) {
        do {
```

```
            swapped = 0;
            ptr1 = &(my_head->first);

            while ((*ptr1)->next != lptr) {
                /* if current element is greater than next -> we swap them */
                if ((strcmp(desc, "id") == 0) ?
                    (*ptr1)->id > (*ptr1)->next->id :
                    (*ptr1)->friendsCount < (*ptr1)->next->friendsCount) {
                    swapNodes(ptr1, *ptr1, (*ptr1)->next);
                    swapped = 1;
                }
                /* getting next element (if current was greater -> we will get it again) */
                ptr1 = &((*ptr1)->next);
            }
            lptr = *ptr1;
        } while (swapped);

        lptr = my_head->first;
        while (lptr != NULL && lptr->next != NULL) {
            lptr = lptr->next;
        }
        my_head->last = lptr;
    }
}

/**
 * Swaps two nodes in a linked list.
 * This function updates the previous node's next pointer to point to the second node, effectively
 * swapping the two nodes in the list. It makes use of a double pointer to the previous node's next
 * field (**prevNode) to directly modify the link between nodes, facilitating the swap operation
 * without additional steps to handle special cases, such as swapping the head of the list.
 *
 * @param prevNode A double pointer to the previous node's next field, pointing to the first of
 *                 the two nodes to be swapped.
 * @param a A pointer to the first node to be swapped.
 * @param b A pointer to the second node to be swapped, immediately following the first.
 */
void swapNodes(User **prevNode, User *a, User *b) {
    (*prevNode) = b;
    a->next = b->next;
```

```
    b->next = a;
}

int startsWithIgnoreCase(const char *str, const char *prefix) {
    int isPrefix = 1;

    while (*str && *prefix && isPrefix) {
        if (tolower(*str) != tolower(*prefix)) {
            isPrefix = 0;
        }
        str++;
        prefix++;
    }
    if (*prefix != '\0') {
        isPrefix = 0;
    }

    return isPrefix;
}

void filterList(Head* head) {
    User* q = NULL;
    User* q1 = NULL;
    char ask;
    int j;
    char temp[MAXLEN];

    printf("You can sort users by either name or profession. Choose one option (1 or 2): ");
    scanf("%c", &ask);
    getchar();
    if (ask != '1' && ask != '2') {
        printf("invalid option");
    } else {
        clearConsole();
        if (ask == '1') {
            printf("Enter the user name: ");
        } else {
            printf("Enter the profession: ");
        }
        scanf("%s", temp);
```

```
        getchar();
        printf("\n");
        printHeader();
        j = 0;
        q = head->first;
        while (q != NULL) {
            if (startsWithIgnoreCase((ask == '2') ? q->profession : q->fullName, temp)) {
                printUser(q);
                j++;
            }
            q1 = q->next;
            q = q1;
        }
        if (j == 0) {
            printf("\nNo user seems to match your input.\n");
        }
    }
}

void clearList(Head* head) {
    User* q;
    User* q1;
    q = head->first;
    while (q != NULL) {
        q1 = q->next;
        deleteNode(head, q);
        q = q1;
    }
}

char **simpleSplit(char *str, int length) {
    int count = 0;
    int i = 0;
    int start = 0;
    int j = 0;
    int wordLen = 0;
    char **result = NULL;
    char *newStr = NULL;
    int allocError = 0;
```

```
    for (i = 0; i < length; i++) {
        if (str[i] == ';') count++;
    }
    count++;

    result = malloc(count * sizeof(char *));
    if (result == NULL) {
        perror("Memory allocation failed");
    } else {
        for (i = 0; i < length; i++) {
            if (str[i] == ';' || str[i] == '\0') {
                wordLen = i - start;
                newStr = malloc((wordLen + 1) * sizeof(char));
                if (newStr == NULL) {
                    perror("Memory allocation failed");
                    allocError = 1;
                    i = length;
                } else {
                    strncpy(newStr, str + start, wordLen);
                    newStr[wordLen] = '\0';
                    result[j++] = newStr;
                    start = i + 1;
                }
            }
        }

        if (allocError) {
            for (i = 0; i < j; i++) {
                free(result[i]);
            }
            free(result);
            result = NULL;
        }
    }

    return result;
}

void simpleSplitInt(User* user, const char *str, int isManual, int idList[], int usersCount) {
    int count = 0, cnt = 0;
```

```
    int start = 0;
    int i, len, flag, n, j;
    char tempStr[MAXLEN];

    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] == ',') cnt++;
    }
    cnt++;

    for (i = 0; i < user->friendsCount; i++) {
        user->friendsId[i] = 0;
    }

    flag = 1;

    for (i = 0; str[i] != '\0' && flag; i++) {
        if (str[i] == ',' || str[i + 1] == '\0') {
            len = (str[i] == ',') ? (i - start) : (i - start + 1);
            strncpy(tempStr, str + start, len);
            tempStr[len] = '\0';

            n = atoi(tempStr);

            flag = isManual ? 0 : 1;
            for (j = 0; j < usersCount && !flag; j++) {
                if (n == idList[j]) {
                    flag = 1;
                }
            }

            user->friendsId[count++] = n;

            start = i + 1;
        }
    }

    if (!flag) {
        puts("It seems that among friends there is a user ID that is not in the database -> setting 0
friends automatically");
        user->friendsCount = 0;
    }
```

```
    if (count < user->friendsCount) {
        printf("It seems that the number of entered IDs does not correspond to the specified number of friends -> updating friends count: %d\n", count);
        user->friendsCount = count;
    }
}

void trim(char *str) {
    int i = 0;

    for (i = 0; i < MAXLEN; i++) {
        if (str[i] == '\n') {
            str[i] = '\0';
            i = MAXLEN;
        }
    }
}

void printHeader() {
    printf("%-3s %-20s %-5s %-15s %-15s %-15s %-15s %-20s\n",
           "ID", "Full Name", "Age", "Profession", "Friends Rating", "Public Rating", "Friends Count", "Friends IDs");
}

void printUser(User *user) {
    int i;
    printf("%-3d %-20s %-5d %-15s %-15.1f %-15.1f %-15d ",
           user->id, user->fullName, user->age, user->profession, user->friendsRating, user->publicRating, user->friendsCount);

    printf("[");
    for (i = 0; i < user->friendsCount; i++) {
```

```
        printf("%d", user->friendsId[i]);
        if (i < user->friendsCount - 1) {
            printf(", ");
        }
    }
    printf("]\n");
}

void printAllUsers(Head* my_head) {
    User *q;
    printHeader();
    q = my_head->first;
    while (q != NULL) {
        printUser(q);
        q = q->next;
    }
}

void pressEnterToContinue() {
    printf("\nPress ENTER to continue ");
    getchar();
    clearConsole();
}

void clearConsole() {
    #if defined(_WIN32) || defined(_WIN64)
        system("cls");
    #else
        system("clear");
    #endif
}
```

Контрольные примеры

Пример:

```
Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit


Option: 1
```

ID	Full Name	Age	Profession	Friends Rating	Public Rating	Friends Count	Friends IDs
1	John Doe	30	teacher	4.5	3.9	3	[2, 5, 7]
2	Jane Smith	25	engineer	3.8	4.1	2	[1, 3]
3	Alice Johnson	28	driver	4.2	3.7	4	[1, 2, 6, 8]
4	Michael Brown	33	pilot	3.9	4.0	5	[3, 6, 9, 10, 2]
5	Emily Davis	27	dentist	4.1	3.8	3	[1, 2, 3]
6	David Wilson	35	actor	4.0	4.2	2	[5, 2]
7	Linda Martinez	32	actor	3.9	3.7	4	[4, 6, 5, 1]
8	Robert White	29	teacher	4.3	3.8	3	[1, 2, 3]
9	Sarah Taylor	31	teacher	4.0	4.1	5	[8, 5, 6, 3, 1]
10	James Anderson	34	pilot	4.2	3.9	2	[1, 2]
11	Davidios Morgan	20	teacher	2.0	1.0	0	[]
12	Casey Taylor	28	pilot	3.7	1.7	5	[6, 8, 13, 10, 15]
13	Jamie Jones	27	dentist	4.4	1.3	5	[8, 3, 11, 1, 7]
14	Charlie Williams	26	engineer	3.4	3.5	0	[]
15	Sam Jones	26	driver	2.8	4.4	3	[3, 5, 2]
16	Jordan Miller	35	engineer	3.0	2.0	4	[1, 13, 4, 14]
17	Chris Moore	20	pilot	2.7	4.4	0	[]
18	Charlie Williams	32	actor	3.1	2.9	1	[14]
19	Casey Wilson	20	dentist	3.1	2.7	0	[]

```
20  Taylor Johnson       32    driver          4.1             2.8             4               [12, 7, 20, 10]
21  Cameron Moore        28    pilot           2.4             2.1             2               [7, 10]
22  Casey Moore          35    actor           3.9             1.8             2               [13, 15]
23  Charlie Moore        24    teacher         3.4             1.7             5               [10, 3, 25, 18,
22]
24  Chris Wilson         27    teacher         4.4             3.6             3               [7, 14, 20]
25  Cameron Wilson       32    dentist         4.3             2.6             5               [8, 14, 15, 23,
29]
26  Chris Johnson        21    pilot           2.5             1.5             0               []
27  Jamie Moore          27    dentist         2.9             4.4             5               [25, 1, 11, 31,
27]
28  Jamie Moore          24    driver          2.2             1.1             0               []
29  Jordan Jones         31    engineer        3.9             3.7             2               [27, 16]
30  Sam Wilson           26    driver          2.3             4.3             0               []

Press ENTER to continue



Choose option and press ENTER:
1. Print all users
2. Sort users by friends count
3. Sort users by id
4. Add new user
5. Filter users by name or profession
6. Delete user before specified id
7. Clear list
8. Exit

Option: 2
```

```
Sorted!


Press ENTER to continue




Choose option and press ENTER:
1. Print all users
2. Sort users by friends count
3. Sort users by id
4. Add new user
5. Filter users by name or profession
6. Delete user before specified id
7. Clear list
8. Exit

Option: 4




Enter full name: new User

Enter age: 12

Enter profession: pilot

Enter friends rating (float number less than 5): 4.23

Enter public rating (float number less than 5): 4.1

Enter friends count (less than 30): 12

Enter friends IDs (example: 1,2,3,4)
Available IDs: [4,9,12,13,23,25,27,3,7,16,20,1,5,8,15,24,2,6,10,21,22,29,18,11,14,17,19,26,28,30]

IDs: 1,2,3
```

It seems that the number of entered IDs does not correspond to the specified number of friends -> updating friends count: 3

New user successfully added!

Press ENTER to continue

Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit

Option: 6

Enter the users id to remove the user BEFORE it

(enter 0 if you don't want to do it)

Here is list of ids:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

Example:

id: 2 -> user with id 1 will be removed

id: 2

Success: user with id 1 has been removed!

```
Press ENTER to continue



Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit


Option: 6



Enter the users id to remove the user BEFORE it

(enter 0 if you don't want to do it)


Here is list of ids:

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]


Example:

id: 2 -> user with id 1 will be removed


id: 2

Failed: there is no user with id 1.


Press ENTER to continue
```

Примеры выполнения программы

```
Choose option and press ENTER:
1. Print all users
2. Sort users by friends count
3. Sort users by id
4. Add new user
5. Filter users by name or profession
6. Delete user before specified id
7. Clear list
8. Exit

Option: 1
```

ID	Full Name	Age	Profession	Friends Rating	Public Rating	Friends Count	Friends IDs
1	John Doe	30	teacher	4.5	3.9	3	[2, 5, 7]
2	Jane Smith	25	engineer	3.8	4.1	2	[1, 3]
3	Alice Johnson	28	driver	4.2	3.7	4	[1, 2, 6, 8]
4	Michael Brown	33	pilot	3.9	4.0	5	[3, 6, 9, 10, 2]
5	Emily Davis	27	dentist	4.1	3.8	3	[1, 2, 3]
6	David Wilson	35	actor	4.0	4.2	2	[5, 2]
7	Linda Martinez	32	actor	3.9	3.7	4	[4, 6, 5, 1]
8	Robert White	29	teacher	4.3	3.8	3	[1, 2, 3]
9	Sarah Taylor	31	teacher	4.0	4.1	5	[8, 5, 6, 3, 1]
10	James Anderson	34	pilot	4.2	3.9	2	[1, 2]
11	Davidios Morgan	20	teacher	2.0	1.0	0	[]
12	Casey Taylor	28	pilot	3.7	1.7	5	[6, 8, 13, 10, 15]
13	Jamie Jones	27	dentist	4.4	1.3	5	[8, 3, 11, 1, 7]
14	Charlie Williams	26	engineer	3.4	3.5	0	[]
15	Sam Jones	26	driver	2.8	4.4	3	[3, 5, 2]
16	Jordan Miller	35	engineer	3.0	2.0	4	[1, 13, 4, 14]
17	Chris Moore	20	pilot	2.7	4.4	0	[]
18	Charlie Williams	32	actor	3.1	2.9	1	[14]
19	Casey Wilson	20	dentist	3.1	2.7	0	[]
20	Taylor Johnson	32	driver	4.1	2.8	4	[12, 7, 20, 10]
21	Cameron Moore	28	pilot	2.4	2.1	2	[7, 10]
22	Casey Moore	35	actor	3.9	1.8	2	[13, 15]
23	Charlie Moore	24	teacher	3.4	1.7	5	[10, 3, 25, 18, 22]
24	Chris Wilson	27	teacher	4.4	3.6	3	[7, 14, 20]
25	Cameron Wilson	32	dentist	4.3	2.6	5	[8, 14, 15, 23, 29]
26	Chris Johnson	21	pilot	2.5	1.5	0	[]
27	Jamie Moore	27	dentist	2.9	4.4	5	[25, 1, 11, 31, 27]
28	Jamie Moore	24	driver	2.2	1.1	0	[]
29	Jordan Jones	31	engineer	3.9	3.7	2	[27, 16]
30	Sam Wilson	26	driver	2.3	4.3	0	[]

```
Press ENTER to continue
```

```
Enter the users id to remove the user BEFORE it
(enter 0 if you don't want to do it)

Here is list of ids:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

Example:
id: 2 -> user with id 1 will be removed

id: 31

Success: user with id 30 has been removed!

Press ENTER to continue
```

```
Enter the users id to remove the user BEFORE it
(enter 0 if you don't want to do it)

Here is list of ids:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]

Example:
id: 2 -> user with id 1 will be removed

id: 33
Failed: there is no user with id 32.

Press ENTER to continue
```

```
Enter the users id to remove the user BEFORE it
(enter 0 if you don't want to do it)

Here is list of ids:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]

Example:
id: 2 -> user with id 1 will be removed

id: 12

Success: user with id 11 has been removed!

Press ENTER to continue
```

ID	Full Name	Age	Profession	Friends Rating	Public Rating	Friends Count	Friends IDs
1	John Doe	30	teacher	4.5	3.9	3	[2, 5, 7]
2	Jane Smith	25	engineer	3.8	4.1	2	[1, 3]
3	Alice Johnson	28	driver	4.2	3.7	4	[1, 2, 6, 8]
4	Michael Brown	33	pilot	3.9	4.0	5	[3, 6, 9, 10, 2]
5	Emily Davis	27	dentist	4.1	3.8	3	[1, 2, 3]
6	David Wilson	35	actor	4.0	4.2	2	[5, 2]
7	Linda Martinez	32	actor	3.9	3.7	4	[4, 6, 5, 1]
8	Robert White	29	teacher	4.3	3.8	3	[1, 2, 3]
9	Sarah Taylor	31	teacher	4.0	4.1	5	[8, 5, 6, 3, 1]
10	James Anderson	34	pilot	4.2	3.9	2	[1, 2]
12	Casey Taylor	28	pilot	3.7	1.7	5	[6, 8, 13, 10, 15]
13	Jamie Jones	27	dentist	4.4	1.3	4	[8, 3, 1, 0]
14	Charlie Williams	26	engineer	3.4	3.5	0	[]
15	Sam Jones	26	driver	2.8	4.4	3	[3, 5, 2]
16	Jordan Miller	35	engineer	3.0	2.0	4	[1, 13, 4, 14]
17	Chris Moore	20	pilot	2.7	4.4	0	[]
18	Charlie Williams	32	actor	3.1	2.9	1	[14]
19	Casey Wilson	20	dentist	3.1	2.7	0	[]
20	Taylor Johnson	32	driver	4.1	2.8	4	[12, 7, 20, 10]
21	Cameron Moore	28	pilot	2.4	2.1	2	[7, 10]
22	Casey Moore	35	actor	3.9	1.8	2	[13, 15]
23	Charlie Moore	24	teacher	3.4	1.7	5	[10, 3, 25, 18, 22]
24	Chris Wilson	27	teacher	4.4	3.6	3	[7, 14, 20]
25	Cameron Wilson	32	dentist	4.3	2.6	5	[8, 14, 15, 23, 29]
26	Chris Johnson	21	pilot	2.5	1.5	0	[]
27	Jamie Moore	27	dentist	2.9	4.4	4	[25, 1, 31, 0]
28	Jamie Moore	24	driver	2.2	1.1	0	[]
29	Jordan Jones	31	engineer	3.9	3.7	2	[27, 16]

```
Press ENTER to continue
```

Выводы.

В ходе выполнения работы были получены практические навыки в работе с линейными односвязными списками в языке С.