Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Программная инженерия»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ № 7

Связь между элементами списков
Тема

Руководитель _____ А.С. Черниговский
Подпись, дата Инициалы, Фамилия

Студент КИ19-17/1Б, №031939174 _____ А.К. Никитин
Номер группы, зачетной книжки Подпись, дата Инициалы, Фамилия

Красноярск 2020

1 Цель

Ознакомиться с типами связей между элементами списков и научится применять их для связывания элементов двух списков.

2 Задачи

Для выполнения лабораторной работы необходимо выполнить следующие задачи:

1) выполнить работу в соответствии с заданием;

2) добавление элементов типа А и Б;

3) создавать связь между элементами типа А и Б;

4) добавить возможность сохранять и загружать данные из файла (в т.ч. связи);

5) выводить на экран все элементы А или Б (на усмотрение пользователя);

6) выводить на экран все элементы А, связанные с выбранным элементом Б и наоборот;

7) добавить возможность удаления выбранного элемента типа А или типа Б;

8) добавить сортировку по одному из выбранных полей.

3 Описание задания

Файловая система. А: пользователь (логин, пароль, реальное имя). Б: файл (имя, размер, дата создания). Пользователь может работать одновременно с несколькими файлами. У каждого файла может быть несколько пользователей.

4 Ход выполнения

Ниже представлен листинг программы по заданию.

Листинг 1 – Реализация связи «Файл-Пользователь»

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

// Тут хранится информация о файле
typedef struct
{
    char* login;
    char* password;
    char* name;
}userInfo;

// А тут о пользователе
typedef struct
{
    char* name;
    int size;
    int date;
}fileInfo;

// Вершина списка всех файлов
typedef struct _list1
{
    fileInfo data;
    struct _list1* next;
    struct _list1* prev;
    struct _list2** connection; // Массив связей
    int connectionLen;  // Его длина
} ListFile;

// Вершина списка всех пользователей
typedef struct _list2
{
    userInfo data;
    struct _list2* next;
    struct _list2* prev;
```

Продолжение листинга

```
    struct _list1** connection;
    int connectionLen;
} ListUser;

// Ввод целого числа
void inputNat(int* number)
{
    while (!scanf("%d", number))
    {
        fflush(stdin);
        printf("Введите корректные данные!\n");
    }
    if (*number < 0)
    {
        printf("Число должно быть натуральным или 0!\n");
        inputNat(number);
    }
}

// Ввод строки произвольной длины
void inputString(char** word)
{
    int count = 0;
    char inputChar = 0;

    fflush(stdin);
    *word = NULL;

    while(1)
    {
        inputChar = getchar();
        if (inputChar == '\n')
            break;
        else
        {
            *word = realloc(*word, count + 1);
            (*word)[count] = inputChar;
            count++;
        }
    }
    (*word)[count] = '\0';
```

Продолжение листинга

```
}

// Добавление нового элемента в двусвязный список
void addFile(ListFile** head, ListFile** tail, fileInfo data)
{
    ListFile* array = (ListFile*)malloc(sizeof(ListFile));
    array->next = NULL;
    array->prev = NULL;
    array->connection = NULL;
    array->connectionLen = 0;
    array->data = data;

    if(*head == NULL)
        *head = *tail = array;
    else {
        (*tail)->next = array;
        array->prev = *tail;
        *tail  = array;
    }
}

void addUser(ListUser** head, ListUser** tail, userInfo data)
{
    ListUser* array = (ListUser*)malloc(sizeof(ListUser));
    array->next = NULL;
    array->prev = NULL;
    array->connection = NULL;
    array->connectionLen = 0;
    array->data = data;

    if(*head == NULL)
        *head = *tail = array;
    else {
        (*tail)->next = array;
        array->prev = *tail;
        *tail  = array;
    }
}

// Функция возвращает элемент списка по его порядковому номеру
ListFile* extractFile(ListFile* head, int position)
```

Продолжение листинга

```
{
    int count = 0;

    if (position < 0)
        return NULL;

    if (head == NULL)
        return NULL;

    while (1)
    {
        count++;
        if (count == position)
            break;
        head = head->next;
    }
    return head;
}

// Функция возвращает элемент списка по его порядковому номеру
ListUser* extractUser(ListUser* head, int position)
{
    int count = 0;
    ListUser* temp = head;

    if (position < 0)
        return NULL;

    if (head == NULL)
        return NULL;

    while (1)
    {
        count++;
        if (count == position)
            break;
        temp = temp->next;
    }
    return temp;
}
```

Продолжение листинга

```
int knowNumberFile(ListFile* head, ListFile* node)
{
    int position = 0;

    for (ListFile* i = head; i != NULL; i = i->next)
    {
        position++;
        if (i == node)
            return position;
    }

    return 0;
}

int knowNumberUser(ListUser* head, ListUser* node)
{
    int position = 0;

    for (ListUser* i = head; i != NULL; i = i->next)
    {
        position++;
        if (i == node)
            return position;
    }

    return 0;
}

// Выводит информацию, хранящуюся в списке файлов
void fileOutput(ListFile* head)
{
    int count = 0;
    fileInfo data;
    for(ListFile* i = head; i != NULL; i = i->next)
    {
        data = i->data;
        count++;
        printf("%d:  Название: %s; Размер: %d КБ; Дата создания: %d\n",
               count, data.name, data.size, data.date);
    }
}
```

Продолжение листинга

```
// Выводит информацию, хранящуюся в списке пользователей
void userOutput(ListUser* head)
{
    int count = 0;
    userInfo data;
    for(ListUser* i = head; i != NULL; i = i->next)
    {
        data = i->data;
        count++;
        printf("%d:  Логин: %s; Пароль: %s; Реальное имя: %s\n",
               count, data.login, data.password, data.name);
    }
}

// Определение длины списка
int lenFile(ListFile* head)
{
    int count = 0;
    for(ListFile* i = head; i != NULL; i = i->next)
        count++;
    return count;
}

int lenUser(ListUser* head)
{
    int count = 0;
    for(ListUser* i = head; i != NULL; i = i->next)
        count++;
    return count;
}

// Освобождение памяти списков
void listFileFree(ListFile** head, ListFile** tail) {
    ListFile* tmp, *nextElem = *head;
    while(nextElem != NULL) {
        tmp = nextElem;
        nextElem = nextElem->next;
        free(tmp);
    }
    *head = *tail = NULL;
```

Продолжение листинга

```
}

void listUserFree(ListUser** head, ListUser** tail) {
    ListUser* tmp, *nextElem = *head;
    while(nextElem != NULL) {
        tmp = nextElem;
        nextElem = nextElem->next;
        free(tmp);
    }
    *head = *tail = NULL;
}

// Удаление вершини из списков
void deleteFile(ListFile** head, ListFile** tail, int index)
{
    ListFile* elem4Delete;

    elem4Delete = extractFile(*head, index);

    if (lenFile(*head) == 1)
    {
        *head = NULL;
        *tail = NULL;
    }

    else if (elem4Delete==(*head)) // если элемент для удаления первый
    {
        (*head) = (*head)->next;
        (*head)->prev = NULL;
        free(elem4Delete);
    }
    else if (elem4Delete==(*tail)) // если элемент для удаления последний
    {
        (*tail) = (*tail)->prev;
        (*tail)->next = NULL;
        free(elem4Delete);
    }
    else  // удаление из середины списка
    {
        elem4Delete->next->prev = elem4Delete->prev;
        if(elem4Delete->next)
```

Продолжение листинга

```
            elem4Delete->prev->next = elem4Delete->next;
        free(elem4Delete);
    }
}

void deleteUser(ListUser** head, ListUser** tail, int index)
{
    ListUser* elem4Delete;

    elem4Delete = extractUser(*head, index);

    if (lenUser(*head) == 1)
    {
        *head = NULL;
        *tail = NULL;
    }

    else if (elem4Delete==(*head)) // если элемент для удаления первый
    {
        (*head) = (*head)->next;
        (*head)->prev = NULL;
        free(elem4Delete);
    }
    else if (elem4Delete==(*tail)) // если элемент для удаления последний
    {
        (*tail) = (*tail)->prev;
        (*tail)->next = NULL;
        free(elem4Delete);
    }
    else  // удаление из середины списка
    {
        elem4Delete->next->prev = elem4Delete->prev;
        if(elem4Delete->next)
            elem4Delete->prev->next = elem4Delete->next;
        free(elem4Delete);
    }
}

// Добавление связи между списками
void addConnectionFile(ListFile** src, ListUser** dest)
{
```

Продолжение листинга

```
    puts("+");
    ListUser** connectionSrc = (*src)->connection;  // Массив связей начала новой
связи
    puts("-");
    ListFile** connectionDest = (*dest)->connection;  // Массив связей конца новой
связи

    int lengthSrc = (*src)->connectionLen;
    int lengthDest = (*dest)->connectionLen;

    for (int i = 0; i < lengthSrc; i++)
        if (connectionSrc[i] == *dest)
            return ;

    connectionSrc = realloc(connectionSrc, sizeof(ListUser*) * ((lengthSrc + 1)));
    connectionSrc[lengthSrc] = *dest;

    (*src)->connection = connectionSrc;
    (*src)->connectionLen++;

    connectionDest = realloc(connectionDest, sizeof(ListFile*) * ((lengthDest +
1)));
    connectionDest[lengthSrc] = *src;

    (*dest)->connection = connectionDest;
    (*dest)->connectionLen++;

    puts("end");
}

void addConnectionUser(ListUser** src, ListFile** dest)
{
    ListFile** connectionSrc = (*src)->connection;
    ListUser** connectionDest = (*dest)->connection;
    int lengthSrc = (*src)->connectionLen;
    int lengthDest = (*dest)->connectionLen;

    printf("%d\n", lengthSrc);
    puts("+");

    for (int i = 0; i < lengthSrc; i++)
```

Продолжение листинга

```
        if (connectionSrc[i] == *dest)
            return ;

    connectionSrc = realloc(connectionSrc, sizeof(ListFile*) * ((lengthSrc + 1)));
    connectionSrc[lengthSrc] = *dest;

    (*src)->connection = connectionSrc;
    (*src)->connectionLen++;

    connectionDest = realloc(connectionDest, sizeof(ListUser*) * ((lengthDest +
1)));
    connectionDest[lengthSrc] = *src;

    (*dest)->connection = connectionDest;
    (*dest)->connectionLen++;

    free(connectionSrc);
    free(connectionDest);
}

// Меняет местами информацию между двумя вершинами
void fileSwap(ListFile** first, ListFile** second)
{
    fileInfo tempData;
    ListUser** tempConn;
    int tempLen;

    tempData = (*first)->data;
    (*first)->data = (*second)->data;
    (*second)->data = tempData;

    tempConn = (*first)->connection;
    (*first)->connection = (*second)->connection;
    (*second)->connection = tempConn;

    tempLen = (*first)->connectionLen;
    (*first)->connectionLen = (*second)->connectionLen;
    (*second)->connectionLen = tempLen;

    free(tempConn);
}
```

Продолжение листинга

```
void userSwap(ListUser** first, ListUser** second)
{
    userInfo tempData;
    ListFile** tempConn;
    int tempLen;

    tempData = (*first)->data;
    (*first)->data = (*second)->data;
    (*second)->data = tempData;

    tempConn = (*first)->connection;
    (*first)->connection = (*second)->connection;
    (*second)->connection = tempConn;

    tempLen = (*first)->connectionLen;
    (*first)->connectionLen = (*second)->connectionLen;
    (*second)->connectionLen = tempLen;

    free(tempConn);
}

// Сортирует списки по дному из полей в структуре
void sortFile(ListFile** head, char* key, int order)
{
    ListFile* temp = NULL;
    if (strcmp(key, "size") == 0)
    {
        // Сортировка пузырьком
        for (ListFile* i = (*head); i != NULL; i = i->next)
            for (ListFile* j = (*head); j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (i->data.size < j->data.size)
                        fileSwap(&i, &i->prev);
                }
                else  // По убыванию
                if (i->data.size > j->data.size)
                    fileSwap(&i, &i->next);

    }
```

Продолжение листинга

```
    else if (strcmp(key, "date") == 0)
    {
        // Сортировка пузырьком
        for (ListFile* i = (*head); i != NULL; i = i->next)
            for (ListFile* j = (*head); j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (i->data.date < j->data.date)
                        fileSwap(&i, &i->prev);
                }
                else  // По убыванию
                    if (i->data.date > j->data.date)
                        fileSwap(&i, &i->next);

    }

    else if (strcmp(key, "name") == 0)
    {
        // Сортировка пузырьком
        for (ListFile* i = (*head); i != NULL; i = i->next)
            for (ListFile* j = i->next; j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (strcmp(i->data.name, j->data.name) > 0)
                        fileSwap(&i, &j);
                }
                else  // По убыванию
                    if (strcmp(i->data.name, j->data.name) < 0)
                        fileSwap(&i, &j);
    }
}

void sortUser(ListUser** head, char* key, int order)
{
    ListUser* temp = NULL;
    if (strcmp(key, "name") == 0)
    {
        // Сортировка пузырьком
        for (ListUser* i = (*head); i != NULL; i = i->next)
            for (ListUser* j = i->next; j != NULL; j = j->next)
                if (order)  // По возрастанию
```

Продолжение листинга

```
                {
                    if (strcmp(i->data.name, j->data.name) > 0)
                        userSwap(&i, &j);
                }
                else  // По убыванию
                if (strcmp(i->data.name, j->data.name) < 0)
                    userSwap(&i, &j);
    }

    if (strcmp(key, "login") == 0)
    {
        // Сортировка пузырьком
        for (ListUser* i = (*head); i != NULL; i = i->next)
            for (ListUser* j = i->next; j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (strcmp(i->data.login, j->data.login) > 0)
                        userSwap(&i, &j);
                }
                else  // По убыванию
                if (strcmp(i->data.login, j->data.login) < 0)
                    userSwap(&i, &j);
    }

    if (strcmp(key, "password") == 0)
    {
        // Сортировка пузырьком
        for (ListUser* i = (*head); i != NULL; i = i->next)
            for (ListUser* j = i->next; j != NULL; j = j->next)
                if (order)  // По возрастанию
                {
                    if (strcmp(i->data.password, j->data.password) > 0)
                        userSwap(&i, &j);
                }
                else  // По убыванию
                if (strcmp(i->data.password, j->data.password) < 0)
                    userSwap(&i, &j);
    }

}
```

Продолжение листинга

```
void putString(char** string, FILE* file)
{
    for (int j = 0; j < strlen(*string); j++)
    {
        fputc(*string[j], file);
    }
    fputc(';', file);
}

void getString(char** string, FILE* file)
{
    int symbol = 0;
    char *word = NULL;
    int count = 0;

    while (!feof(file))
    {
        symbol = fgetc(file);
        if (symbol == -1)
        {
            *string = NULL;
            return ;
        }

        if ((char) symbol == ';')
            break;
        word = realloc(word, count + 1);
        word[count] = (char) symbol;
        count++;
    }
    *string = word;
}

void putFileNode(fileInfo data, FILE* file)
{
    fputs(data.name, file);
    fputc(';', file);
    fwrite(&data.date, sizeof(int), 1, file);
    fwrite(&data.size, sizeof(int), 1, file);
}
```

Продолжение листинга

```
void putUserNode(userInfo data, FILE* file)
{
    fputs(data.name, file);
    fputc(';', file);
    fputs(data.login, file);
    fputc(';', file);
    fputs(data.password, file);
    fputc(';', file);
}

int main()
{
    FILE *dbFile;
    FILE *dbUser;
    FILE *connFile;

    int userChoice = 0;

    ListFile* headFile = NULL, *tailFile = NULL;
    ListUser* headUser = NULL, *tailUser = NULL;

    setlocale(LC_ALL, "");

    enum Case {fileInput=1, userInput, output, file2users, user2files,
showConnection,
               loadData, saveData, delete,  sort, exitProg};

    do
    {
        puts("Введите ваш выбор:\n"
             "1. Добавить файл\n"
             "2. Добавить пользователя\n"
             "3. Вывод информации на экран\n"
             "4. Связать файл с пользователями\n"
             "5. Связать пользователя с файлами\n"
             "6. Вывести связи у выбранного элемента\n"
             "7. Загрузить информацию из файла\n"
             "8. Сохранить информацию в файл\n"
             "9. Удалить элемент из списков\n"
             "10. Отсортировать списки\n"
             "11. Выйти из программы\n");
```

Продолжение листинга

```
        inputNat(&userChoice);

        if (userChoice < 1 || userChoice > 13)
        {
            printf("Введите значение от 1 до 11!\n");
            continue;
        }

        switch (userChoice)
        {
            // Ввод одного файла и присоединение его к списку
            case (fileInput):
            {
                fileInfo info;
                puts("Введите имя файла:");
                inputString(&info.name);
                puts("Введите размер файла:");
                inputNat(&info.size);
                puts("Введите дату создания файла (в секундах):");
                inputNat(&info.date);

                addFile(&headFile, &tailFile, info);
                break;
            }

            // Ввод одного пользователя и присоединение его к списку
            case (userInput):
            {
                userInfo info;
                puts("Введите логин:");
                inputString(&info.login);
                puts("Введите пароль:");
                inputString(&info.password);
                puts("Введите реальное имя:");
                inputString(&info.name);

                addUser(&headUser, &tailUser, info);
                break;
            }

            // Вывод информации из списков
```

Продолжение листинга

```
case (output):
{
    puts("Какую информацию вы хотите вывести на экран?\n");
    puts("1. Информация о всех файлах;\n"
         "2. Информация о всех пользователях;\n"
         "3. Информация о всех файлах и пользователях.\n");
    inputNat(&userChoice);

    if (userChoice < 1 || userChoice > 3)
    {
        printf("Введите значение от 1 до 3!\n");
        break;
    }

    if (userChoice == 1)
    {
        if (headFile != NULL)
            fileOutput(headFile);
        else
            puts("База данных пуста.");
    }
    if (userChoice == 2)
    {
        if (headUser != NULL)
            userOutput(headUser);
        else
            puts("База данных пуста.");
    }
    if (userChoice == 3)
    {
        puts("ИНФОРМАЦИЯ О ФАЙЛАХ:\n");
        if (headFile != NULL)
            fileOutput(headFile);
        else
            puts("База данных пуста.");

        puts("ИНФОРМАЦИЯ О ПОЛЬЗОВАТЕЛЯХ:\n");
        if (headUser != NULL)
            userOutput(headUser);
        else
            puts("База данных пуста.");
```

Продолжение листинга

```
            }
            break;
        }

        // Создание связей у выбранного файла с пользователями
        case (file2users):
        {
            ListFile* sourceFile = NULL;
            ListUser* destinationUser = NULL;

            puts("Выберите файл для работы:");
            fileOutput(headFile);
            inputNat(&userChoice);

            if (userChoice < 1 || userChoice > lenFile(headFile))
            {
                printf("Введите значение от 1 до %d!\n", lenFile(headFile));
                break;
            }

            sourceFile = extractFile(headFile, userChoice);
            while (userChoice != 0)
            {
                puts("Выберите пользователя для создания связи (0 для
выхода):\n");
                userOutput(headUser);
                inputNat(&userChoice);

                if (userChoice == 0)
                    break;

                if (userChoice < 1 || userChoice > lenUser(headUser))
                {
                    printf("Введите значение от 0 до %d!\n",
lenUser(headUser));
                    continue;
                }

                sourceFile = extractFile(headFile, userChoice);  // Файл, из
которого идет связь
```

Продолжение листинга

```
                    destinationUser = extractUser(headUser, userChoice);    //
Файл, в который идет связь
                    addConnectionFile(&sourceFile, &destinationUser);
                }
                break;
            }

            // Создание связей у выбранного пользователя с файлами
            case (user2files):
            {
                ListUser* sourceUser = NULL;
                ListFile* destinationFile = NULL;

                puts("Выберите пользователя для работы:");
                userOutput(headUser);
                inputNat(&userChoice);

                if (userChoice < 1 || userChoice > lenUser(headUser))
                {
                    printf("Введите значение от 1 до %d!\n", lenUser(headUser));
                    break;
                }

                while (userChoice != 0)
                {
                    puts("Выберите файл для создания связи (0 для выхода):\n");
                    fileOutput(headFile);
                    inputNat(&userChoice);

                    if (!userChoice)
                        break;

                    if (userChoice < 1 || userChoice > lenFile(headFile))
                    {
                        printf("Введите    значение    от    0    до    %d!\n",
lenFile(headFile));
                        continue;
                    }

                    sourceUser = extractUser(headUser, userChoice);
                    destinationFile = extractFile(headFile, userChoice);
```

Продолжение листинга

```
                addConnectionUser(&sourceUser, &destinationFile);
            }
            break;
        }

        // Вывод связей у выбранной вершины списка
        case (showConnection):
        {
            int length = 0;
            puts("Вы хотите работать с файлами (1) или пользователями (0)?");
            inputNat(&userChoice);

            if (userChoice == 1)
            {
                userInfo info;
                ListUser** connections = NULL;
                ListFile* currentFile = NULL;

                puts("Введите номер файла:\n");
                fileOutput(headFile);
                inputNat(&userChoice);

                if (userChoice < 1 || userChoice > lenFile(headFile))
                {
                    printf("Введите значение от 1 до %d!\n",
lenFile(headFile));
                    break;
                }

                // Получение информации о файле выбранного номера
                currentFile = extractFile(headFile, userChoice);
                connections = currentFile->connection;
                length = currentFile->connectionLen;

                if (length == 0)
                {
                    puts("Никаких связей нет.");
                    break;
                }

                for (int i = 0; i < length; i++)
```

Продолжение листинга

```
                    {
                        info = connections[i]->data;
                        printf("%s\n", info.name);
                    }
                    free(connections);
                }

                else if (userChoice == 0)
                {
                    fileInfo info;
                    ListFile** connections = NULL;
                    ListUser* currentUser = NULL;

                    puts("Введите номер пользователя:\n");
                    userOutput(headUser);
                    inputNat(&userChoice);

                    if (userChoice < 1 || userChoice > lenUser(headUser))
                    {
                        printf("Введите значение от 1 до %d!\n",
lenUser(headUser));
                        break;
                    }

                    // Получение информации о пользователе выбранного номера
                    currentUser = extractUser(headUser, userChoice);
                    connections = currentUser->connection;
                    length = currentUser->connectionLen;

                    if (length == 0)
                    {
                        puts("Никаких связей нет.");
                        break;
                    }

                    for (int i = 0; i < length; i++)
                    {
                        info = connections[i]->data;
                        printf("%s\n", info.name);
                    }
                    free(connections);
```

Продолжение листинга

```
                }
                else
                    puts("Введите 1 или 0!");
                break;
            }

            // Загрузить информацию из файла в массив
            case (loadData):
            {
                fileInfo info1;
                info1.name = NULL;
                userInfo info2;
                info2.name = NULL;
                int length = 0;
                int count = 0;
                int position;
                int nextSymbol;
                ListUser* connectedUser;
                ListFile* connectedFile;

                dbFile = fopen("databaseFile.txt", "r");

                if (dbFile == NULL)
                {
                    printf("База файлов отсутствует. Она будет создана
автоматически.\n");
                    dbFile = fopen("databaseFile.txt", "w");
                    fclose(dbFile);
                    break;
                }

                listFileFree(&headFile, &tailFile); // Очистка списков для
создания новых

                while (!feof(dbFile))
                {
                    getString(&info1.name, dbFile);
                    if (info1.name == NULL)
                        break;

                    fread(&info1.date, sizeof(int), 1, dbFile);
```

Продолжение листинга

```
                fread(&info1.size, sizeof(int), 1, dbFile);

                addFile(&headFile, &tailFile, info1);  // Построение нового
списка
            }

            fclose(dbFile);

            dbUser = fopen("databaseUser.txt", "r");

            if (dbUser == NULL)
            {
                printf("База файлов отсутствует. Она будет создана
автоматически.\n");
                dbUser = fopen("databaseUser.txt", "w");
                fclose(dbUser);
                break;
            }

            listUserFree(&headUser, &tailUser); // Очистка списков для
создания новых

            while (!feof(dbUser))
            {
                getString(&info2.name, dbUser);
                if (info2.name == NULL)
                    break;

                getString(&info2.login, dbUser);
                getString(&info2.password, dbUser);

                addUser(&headUser, &tailUser, info2);  // Построение нового
списка
            }

            fclose(dbUser);


            // Запись связей
            connFile = fopen("connections.txt", "r");
            while(!feof(connFile))
```

Продолжение листинга

```
        {
            nextSymbol = getc(connFile);
            fseek(connFile, -1, SEEK_CUR);
            if (nextSymbol == '.')
                break;

            count++;
            connectedFile = extractFile(headFile, count);
            fread(&length, sizeof(int), 1, connFile);

            if (length == 0)
                break;

            if (length == -1)
            {
                connectedFile->connection = NULL;
                connectedFile->connectionLen = 0;
            }

            for (int j = 0; j < length; j++)
            {
                fread(&position, sizeof(int), 1, connFile);
                connectedUser = extractUser(headUser, position);
                addConnectionFile(&connectedFile, &connectedUser);
            }
        }

        fclose(connFile);
        break;
    }

    // Загрузить информацию из списка в файл
    case (saveData):
    {
        fileInfo info1;
        userInfo info2;
        int number = 0;
        int length = 0;

        if (headFile == NULL && headUser == NULL)
        {
```

Продолжение листинга

```
            puts("Сначала заполните списки!");
            break;
        }

        dbFile = fopen("databaseFile.txt", "w");
        connFile = fopen("connections.txt", "w");
        for (ListFile* i = headFile; i != NULL; i = i->next)
        {
            info1 = i->data;
            putFileNode(info1, dbFile);

            // Создание связей
            length = i->connectionLen;
            if (length == 0)
                length = -1;
            fwrite(&length, sizeof(int), 1, connFile);

            for (int j = 0; j < i->connectionLen; j++)
            {
                number = knowNumberUser(headUser, i->connection[j]);
                fwrite(&number, sizeof(int), 1, connFile);
            }
        }
        putc('.', connFile);

        fclose(dbFile);
        fclose(connFile);

        dbUser = fopen("databaseUser.txt", "w");
        for (ListUser* i = headUser; i != NULL; i = i->next)
        {
            info2 = i->data;
            putUserNode(info2, dbUser);
        }

        fclose(dbUser);

        printf("Данные были сохранены в файл.\n");
        break;
    }
```

Продолжение листинга

```
            // Удаление одной вершины из списка
            case (delete):
            {
                puts("Вы хотите работать с файлами (1) или пользователями (0)?");
                inputNat(&userChoice);

                if (userChoice == 1)
                {
                    puts("Введите номер файла:\n");
                    fileOutput(headFile);
                    inputNat(&userChoice);

                    if (userChoice < 1 || userChoice > lenFile(headFile))
                    {
                        printf("Введите значение от 1 до %d!\n",
lenFile(headFile));
                        break;
                    }

                    deleteFile(&headFile, &tailFile, userChoice);
                }
                else if (userChoice == 0)
                {
                    puts("Введите номер пользователя:\n");
                    userOutput(headUser);
                    inputNat(&userChoice);

                    if (userChoice < 1 || userChoice > lenUser(headUser))
                    {
                        printf("Введите значение от 1 до %d!\n",
lenUser(headUser));
                        break;
                    }

                    deleteUser(&headUser, &tailUser, userChoice);
                } else
                    puts("Введите 1 или 0!");
                break;
            }
```

Продолжение листинга

```
            // Сортировка списка по одному из полей
            case (sort):
            {
                int order = 0;
                char* field = NULL;

                puts("Вы хотите сортировать список файлов (1) или пользователей
(0)?");
                inputNat(&userChoice);

                if (userChoice != 1 && userChoice != 0)
                {
                    printf("Введите 1 или 0!");
                    break;
                }

                if (userChoice == 1)
                {
                    printf("По какому полю вы хотите произвести сортировку?\n"
                           "Доступные поля: name, size, date:\n");
                    inputString(&field);
                    printf("Вы хотите произвести сортировку по возрастанию или
убыванию?(1/0)\n");
                    inputNat(&order);

                    if (order != 1 && order != 0)
                    {
                        printf("Введите 1 или 0!");
                        break;
                    }

                    sortFile(&headFile, field, order);
                }
                else if (userChoice == 0)
                {
                    printf("По какому полю вы хотите произвести сортировку?\n"
                           "Доступные поля: login, password, name:\n");
                    inputString(&field);
                    printf("Вы хотите произвести сортировку по возрастанию или
убыванию?(1/0)\n");
```

Продолжение листинга

```
                    inputNat(&order);

                    if (order != 1 && order != 0)
                    {
                        printf("Введите 1 или 0!");
                        break;
                    }

                    sortUser(&headUser, field, order);
                }
                break;
            }

            // Выход
            case (exitProg):
            {
                listFileFree(&headFile, &tailFile);
                exit(1);
                break;
            }
        }
    }while (1);
}
```

5 Результат

Ниже представлен скриншот с консольным выводом.

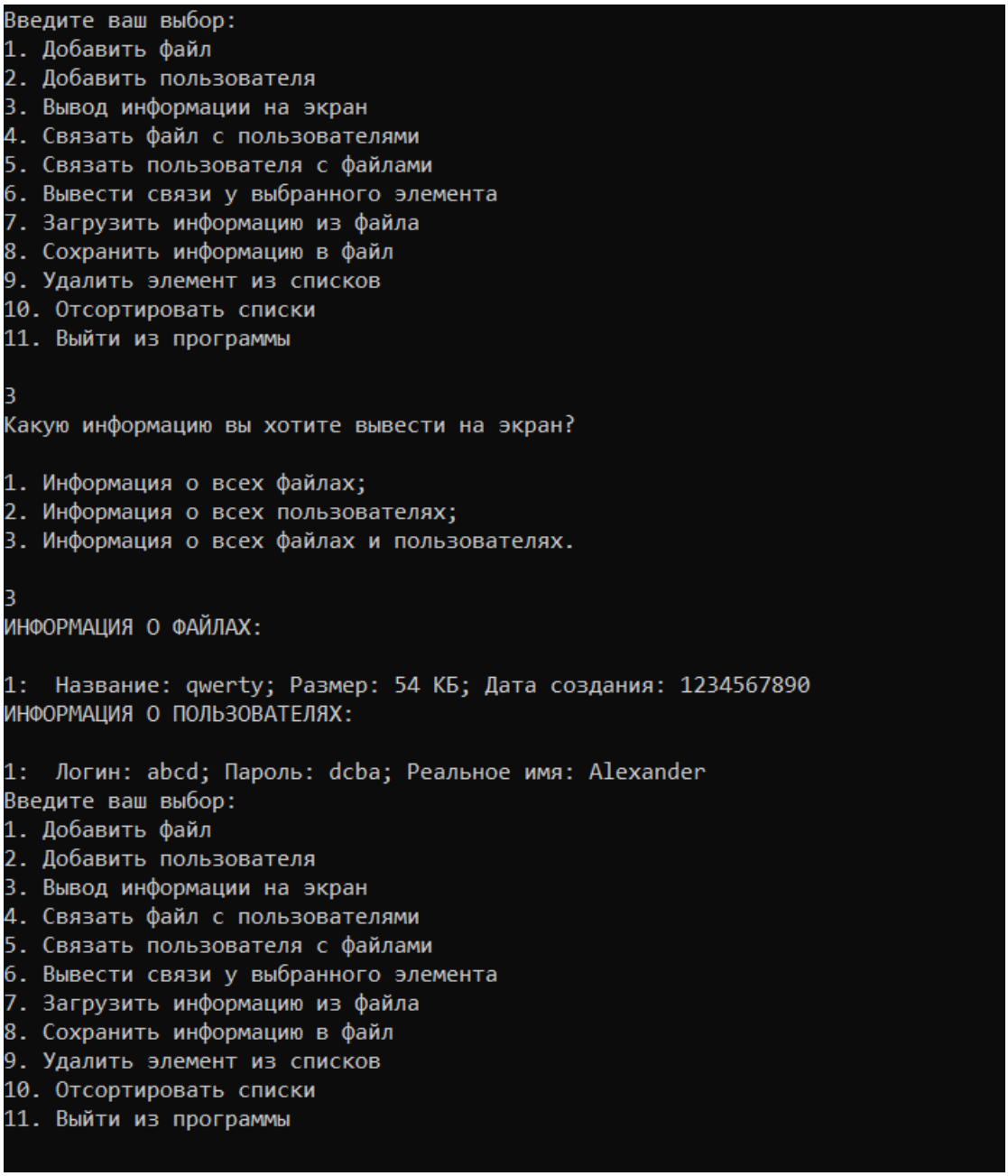

Рисунок 1 – Вывод введенных с клавиатуры данных

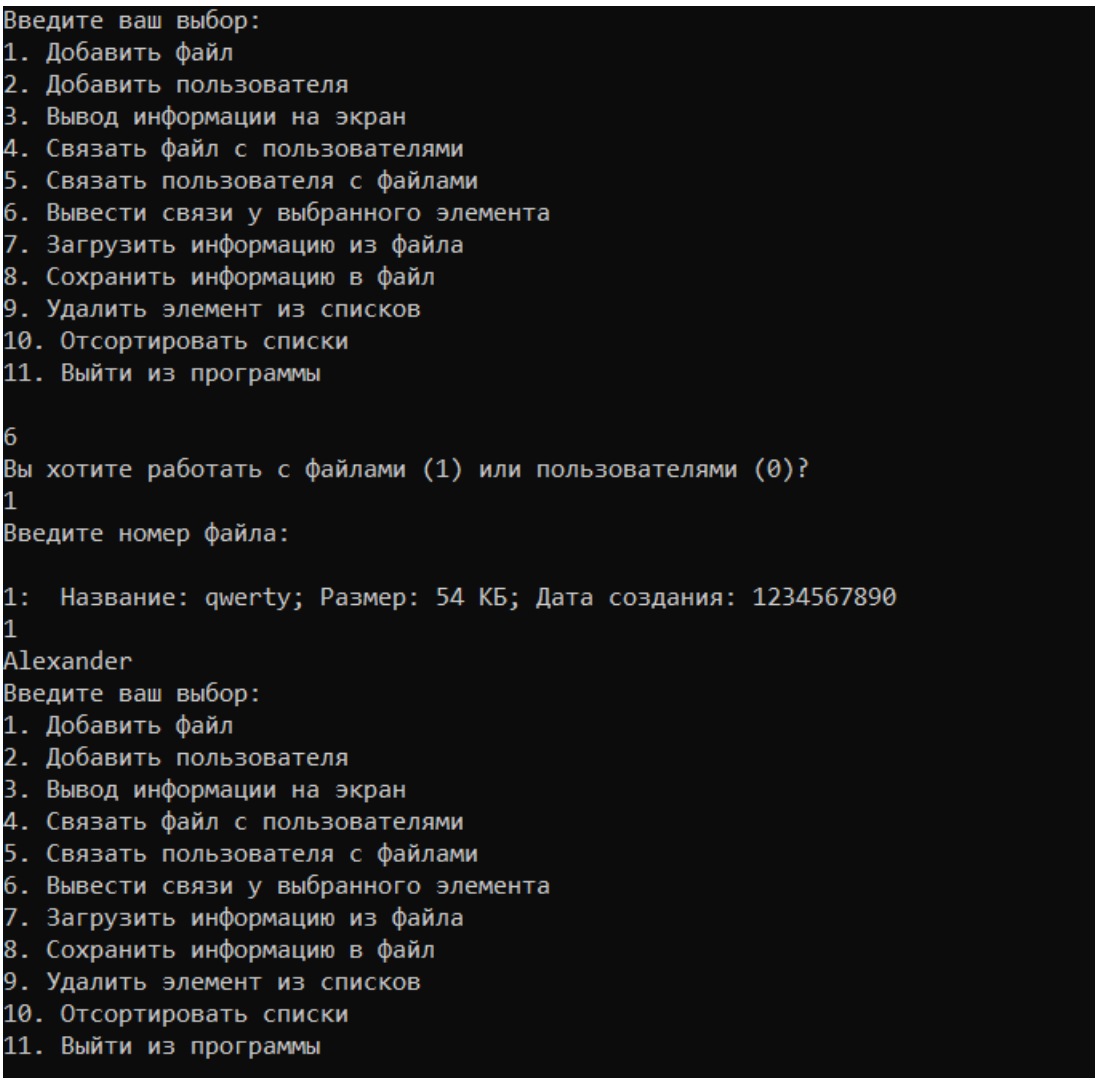

Рисунок 2 – Связывание пользователя и файла

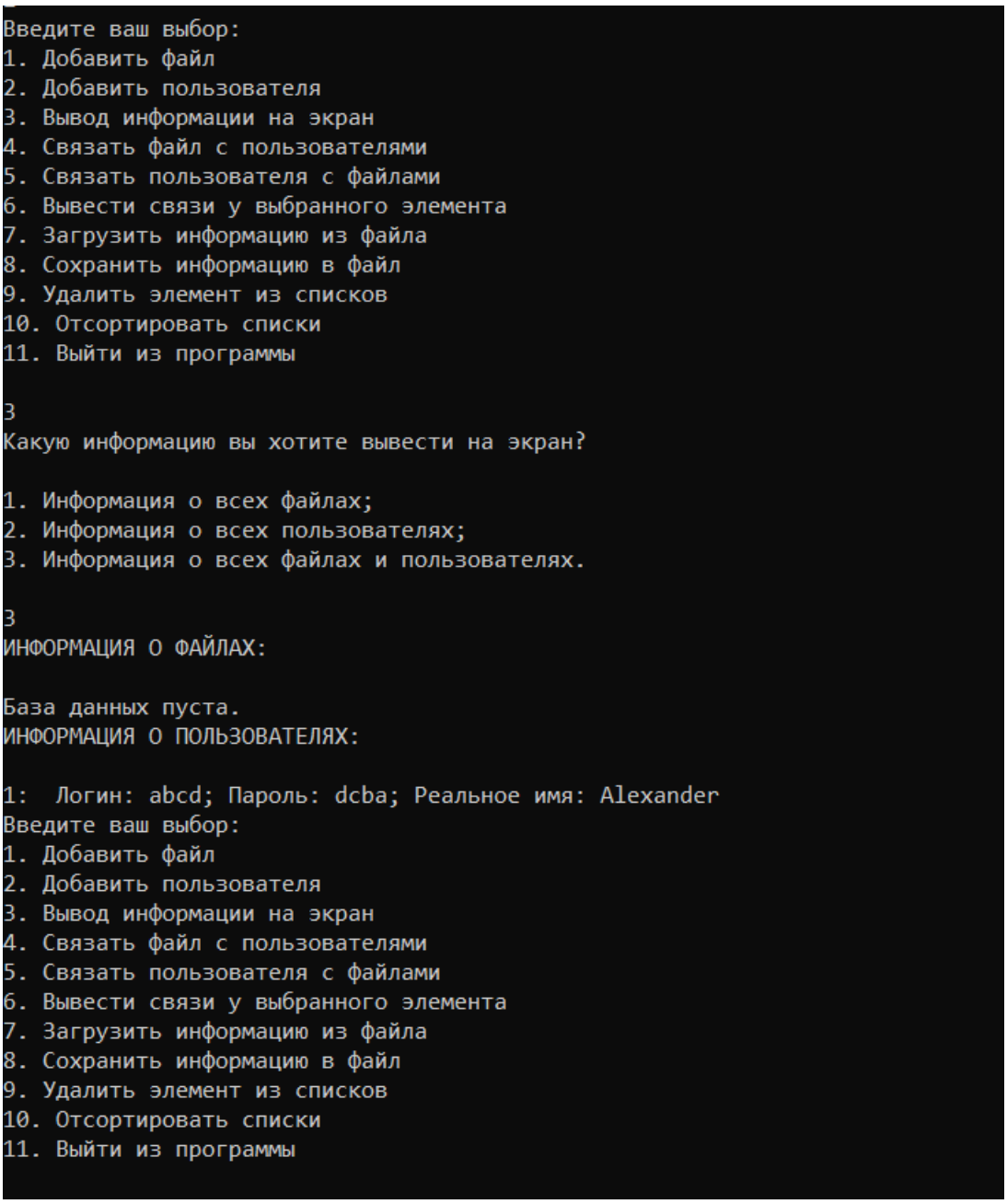

Рисунок 3 – Ручное удаление одной из вершин списка

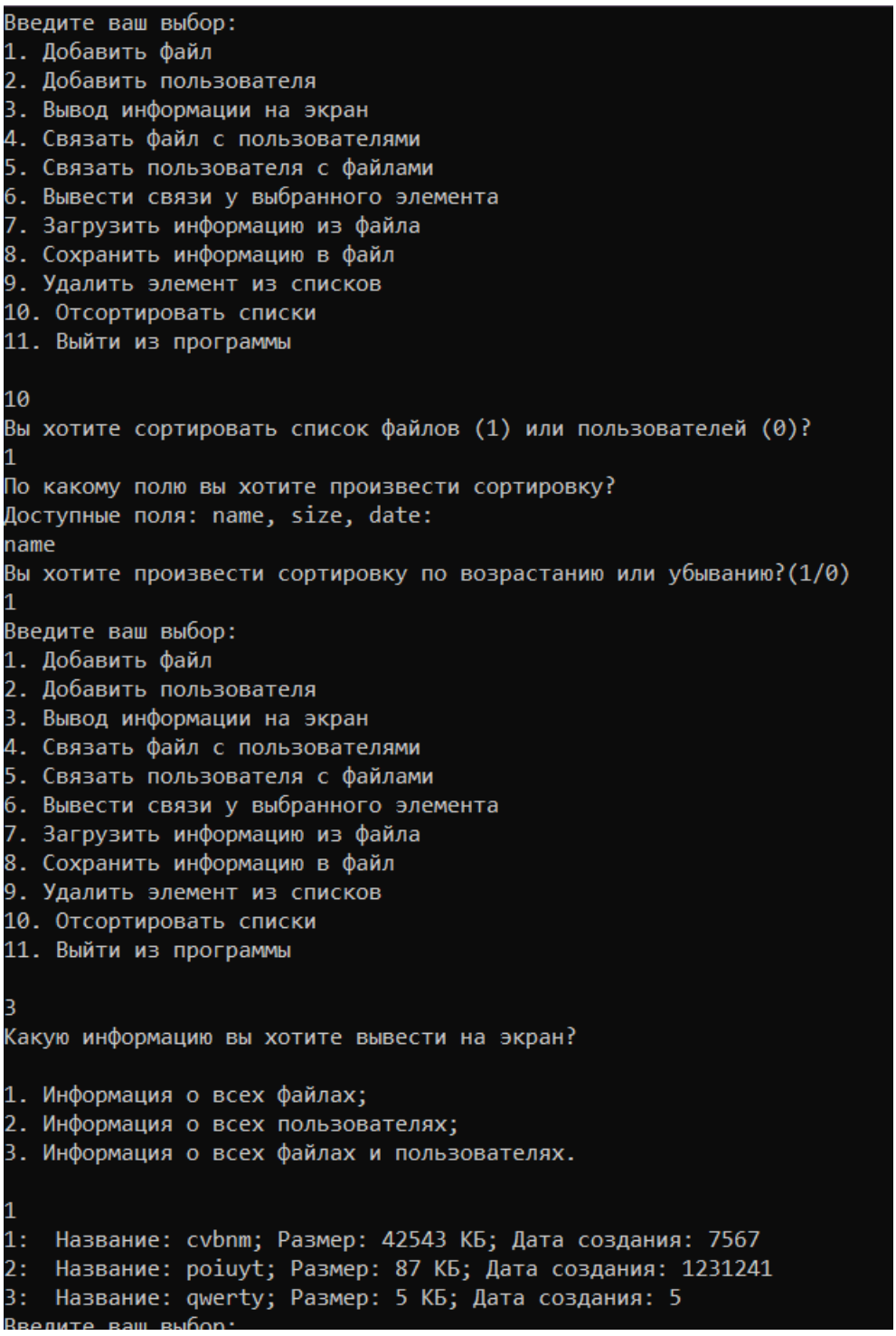

Рисунок 4 – Пример лексикографической сортировки

6 Выводы

По окончании работ были выполнены следующие задачи:

1) изучены основные принципы в работе с несколькими списками;

2) исследованы различные типы связей между списками и реализована связь типа «многие-ко-многим»;

3) реализована программа по заданию.