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

# ОТЧЕТ

**по лабораторной работе №3**

**по дисциплине «Программирование»**

**ТЕМА: РЕКУРСИЯ, ЦИКЛЫ, РЕКУРСИВНЫЙ ОБХОД ФАЙЛОВОЙ СИСТЕМЫ**

Студент гр. 3341 _____ Игнатьев К.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

**Цель работы**

Цель данной работы заключается в изучении и применении рекурсивных функций, а также в освоении работы с файловой системой, включая ее рекурсивное исследование. Для успешного выполнения поставленной цели необходимо выполнить ряд задач:

-Познакомиться с концепцией рекурсии;

-Научиться создавать рекурсивные функции на языке Си;

-Изучить методы взаимодействия с файловой системой на языке Си;

-Разработать программу для рекурсивного просмотра файлов в директории, включая поиск в поддиректориях.

**Задание**

Дана некоторая корневая директория, в которой может находиться некоторое количество папок, в том числе вложенных. В этих папках хранятся некоторые текстовые файлы, имеющие имя вида .txt.

Требуется найти файл, который содержит строку "Minotaur" (файл-минотавр).

Файл, с которого следует начинать поиск, всегда называется file.txt (но полный путь к нему неизвестен).

Каждый текстовый файл, кроме искомого, может содержать в себе ссылку на название другого файла (эта ссылка не содержит пути к файлу). Таких ссылок может быть несколько.

Пример:

Содержимое файла a1.txt

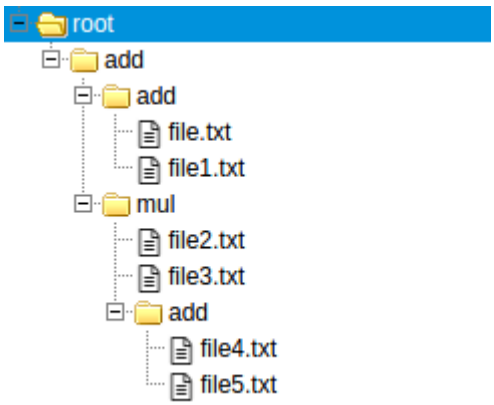
@include a2.txt
@include b5.txt
@include a7.txt

А также файл может содержать тупик:
Содержимое файла a2.txt

Deadlock

Программа должна вывести правильную цепочку файлов (с путями), которая привела к поимке файла-минотавра.

Пример




file.txt:

@include file1.txt

@include file4.txt

@include file5.txt

file1.txt:

Deadlock

file2.txt:

@include file3.txt

file3.txt:

Minotaur

file4.txt:

@include file2.txt

@include file1.txt

file5.txt:

Deadlock

Правильный ответ:

./root/add/add/file.txt

./root/add/mul/add/file4.txt

./root/add/mul/file2.txt

./root/add/mul/file3.txt

Цепочка, приводящая к файлу-минотавру может быть только одна.

Общее количество файлов в каталоге не может быть больше 3000.

Циклических зависимостей быть не может.

Файлы не могут иметь одинаковые имена.

Ваше решение должно находиться в директории /home/box, файл с решением должен называться solution.c. Результат работы программы должен быть записан в файл result.txt. Ваша программа должна обрабатывать директорию, которая называется labyrinth.

**Основные теоретические положения**

1. Язык программирования C предоставляет мощные и гибкие средства для работы с файлами и каталогами, что позволяет программистам создавать сложные приложения, включая те, что требуют обхода файловой системы. Этот процесс зачастую включает в себя рекурсию, метод, при котором функция вызывает саму себя для выполнения задачи. Работа с файловой иерархией в C требует понимания базовых операций с файлами и каталогами, а также умения применять рекурсивные алгоритмы для их эффективного обхода.

2. Файловая система организована в виде иерархии, которая включает в себя каталоги (или папки) и файлы. Каталоги могут содержать другие каталоги и файлы, образуя древовидную структуру. Для работы с этой структурой программа должна быть способна открывать каталоги, читать их содержимое, а также выполнять операции с файлами.

3. Рекурсия — это техника в программировании, при которой функция вызывает саму себя. Она идеально подходит для обработки структур, которые имеют вложенную, иерархическую организацию, как, например, файловая система. Рекурсия позволяет эффективно обходить все каталоги и файлы, начиная с заданной точки, и выполнять необходимые операции (например, поиск, копирование или перемещение).

**Выполнение работы**

1. Функция find_file(const char *dir_name, const char *filename, char** allFiles, int* index, bool* found):

принимает на вход название начальной папки поиска, имя файла, массив с путями к файлам, индекс для заполнения массива и переменную для обозначения нахождения файла. Рекурсивно выполняет поиск файла по папкам, а так же заполняет массив нужными путями.

2. Функция pathcat(const char *path1, const char *path2):

Принимает на вход старый путь и, файл или папку, которые объединяет в один путь и возвращает его.

3. Функция char* getFilename(char* buf) из строки, содержащейся в файле достает название файла, который ищет функция find_file.
4. В функции main() объявляются двумерный динамический массив char** allFiles символов для заполнения путями, переменная index для его корректного заполнения и последующего заполнения файла с результатом работы программы, переменная found типа bool для обозначения, был найден файл-Минотавр или нет. Вызывается функция find_file для начала поиска. Начальным файлом является файл file.txt и папка labyrinth. После завершения работы функции и возврату к функции main создается файл result.txt, в который записываются все данные из массива allFiles, после чего файл закрывается, а массив освобождается.

**Тестирование**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | ./labyrinth | ./labyrinth/add/add/file.txt<br>./labyrinth/add/mul/add/file4.txt<br>./labyrinth/add/mul/file2.txt<br>./labyrinth/add/mul/file3.txt | Тест с сайта e.moevm |
| 2. | ./test | ./test/file.txt<br>./test/pop/file3.txt<br>./test/file4.txt | Дополнительный тест |

**Выводы**

Поставленная цель достигнута, освоена работа с файловой системой, изучены и применены в программе рекурсивные функции для обхода файловой системы. В результате разработана программа для рекурсивного просмотра файлов в директории, включая поиск в поддиректориях, выбирающая файлы с определёнными названиями, которые берутся из отдельных файлов.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: solution.c

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <stdbool.h>


#define MAX_FILE_COUNT 3000
#define ENDLINE '\n'
#define ENDSTR '\0'
#define MAX_STR_SIZE 256

const char* current_dir = ".";
const char* parrent_dir = "..";


char* getFilename(char* buf){
    char* filename = malloc(MAX_STR_SIZE*sizeof(char));
    int index=0;
    int filename_start_index=9;
    for (int i = filename_start_index; i < strlen(buf); ++i) {
        filename[index]=buf[i];
        index++;
    }
    if (filename[index-1]==ENDLINE){
        filename[index-1]=ENDSTR;
    }else {
        filename[index] = ENDSTR;
    }
    return filename;
}

//П о и с к  ф а й л а

char *pathcat(const char *path1, const char *path2){
    int res_path_len = strlen(path1) + strlen(path2) + 2;
    char *res_path = malloc(res_path_len*sizeof(char));
    sprintf(res_path, "%s/%s", path1 ,path2);
    return res_path;
}

void checkFile(bool* found, char** allFiles, char* full_path_file,
int* index);

char *find_file(const char *dir_name, const char *filename, char**
allFiles, int* index, bool* found){
    char *full_path_file = NULL;
    DIR *dir = opendir(dir_name);
    if(dir){
        struct dirent *reading_element = readdir(dir);
```

```
            while (reading_element){
                if(reading_element->d_type          ==          DT_REG
&& !strcmp(reading_element->d_name, filename)){

                    full_path_file = pathcat(dir_name, filename);

                    checkFile(found, allFiles, full_path_file, index);

                }else if (reading_element->d_type == DT_DIR &&
                          strcmp(reading_element->d_name,
current_dir) != 0 &&
                          strcmp(reading_element->d_name,
parrent_dir) != 0){

                    char       *new_dir       =       pathcat(dir_name,
reading_element->d_name);

                    full_path_file   =   find_file(new_dir,   filename,
allFiles, index, found);
                    free(new_dir);
                }
                if(full_path_file)
                    break;
                reading_element = readdir(dir);
            }
            closedir(dir);
        }else
            printf("Failed to open %s directory\n", dir_name);
        return full_path_file;
    }

    void checkFile(bool* found, char** allFiles, char* full_path_file,
int* index){
        FILE *file=fopen(full_path_file, "r");
        char buf[MAX_STR_SIZE];
        if (file){
            while ((fgets(buf, MAX_STR_SIZE, file))!=NULL){
                if (*found== true)
                    break;

                if (strcmp(buf, "Minotaur")==0){
                    allFiles[*index]=full_path_file;
                    (*index)++;
                    *found = true;
                    break;
                }else{
                    char* nameNextFile = getFilename(buf);
                    if (strcmp(nameNextFile, "")==0){
                        nameNextFile=NULL;
                        break;
                    }
                    if (nameNextFile!=NULL){
                        if (*index>0){

                            if               (strcmp(allFiles[*index-1],
full_path_file)!=0){
                                allFiles[*index] = full_path_file;
                                (*index)++;
```

```
                                }
                                for (int i = 0; i < *index-1; ++i) {
                                    if(strcmp(allFiles[i],
full_path_file)==0){
                                        for (int j = i++; j < (*index)-1;
++j) {
                                            allFiles[j]="NULL";
                                        }
                                    }
                                }
                            }else{
                                allFiles[*index] = full_path_file;
                                (*index)++;
                            }
                            find_file("./test",   nameNextFile,   allFiles,
index, found);
                        }
                    }
                }
                fclose(file);
            }
        }

        void createResultFile(int index, char** allFiles){
            FILE *res = fopen("result.txt", "w");
            for (int i = 0; i<index; ++i){
                if(strcmp(allFiles[i], "NULL")!=0) {
                    fprintf(res, "%s\n", allFiles[i]);
                }
            }
            fclose(res);
        }

        int main(){
            char** allFiles = malloc(MAX_FILE_COUNT*sizeof(char));
            int index=0;
            bool found=false;
            find_file("./test", "file.txt", allFiles, &index, &found);
            createResultFile(index,allFiles);
            free(allFiles);
            return 0;
        }
```