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: ОБХОД ФАЙЛОВОЙ СИСТЕМЫ.

Студентка гр. 1304 _____ Ярусова Т. В.

Преподаватель _____ Чайка К. В.

Санкт-Петербург

2022

Цель работы.

Изучить обход файловой системы, его реализация на языке Си с использованием библиотеки *dirent.h*.

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.

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***.

Выполнение работы.

Описание структуры ***File***:

Структура *File* будет хранить в себе информацию о файле.

Структура состоит из 2-х полей: *char* filename* – указатель на строку, в которой будет содержаться имя файла; *char* path* – указатель на строку, в которой будет содержаться путь к данному файлу.

Чтобы не писать каждый раз “*struct File*”, используется оператор *typedef*.

Описание структуры ***Maze*:**

Структура *Maze* будет хранить пути к файлу-минотавру, т.е. ответ.

Структура состоит из 3-х полей: *char** links* – массив строк-путей к файлу-минотавру; *int count* – текущее количество элементов в массиве; *int maxcount* – число элементов, под которые выделена память.

Чтобы не писать каждый раз “*struct Maze*”, используется оператор *typedef*.

Функции:

int main();

Устанавливается путь обрабатываемой директории и файл, с которого следует начать поиск минотавра согласно условию***:*** *const char* root="./labyrinth"; и char* fname="file.txt";*

Объявим структуру массива, в котором будет храниться ответ *Maze path*. Определим, обращаясь к полям структуры, число элементов массива *path.maxcount=10*, под которое пока что следует выделить память и потом, в случае чего, ее расширять, текущее число элементов в массиве, равное 0, *path.count=0*. И выделим небольшое количество памяти под этот массив с помощью функции *malloc()*.

Вызываем функцию *findMinotaur(startDir, filename, &path);* По результату у нас будет заполненный массив Maze, однако пути цепочки будут записаны в обратном порядке, так как функция *findMinotaur()* рекурсивна.

Откроем файл на запись результата: *FILE* res_file = fopen("result.txt","w"*); С помощью цикла *for* в файл запишутся пути к файлу-минотавру, которые хранятся *path.links*.

После записи ответа в файл *result. txt* с помощью функции *fclose()* закрывается данный файл и с помощью функций *free_memory()* и *free()* происходит освобождение выделенной динамической памяти .

void setPath(const char* pathDir, File* path_file);

В данной функции совершается поиск файла с именем *path_file->filename* и при его нахождении в *path_file->path* записывается путь к данному файлу.

void push_answer(Maze* arr, char* path);

В данной функции происходит запись в структуру *Maze* путей, которые способствуют поимке файла-минотавра.

int findMinotaur(const char* startDir, char* filename, Maze* arr);

Данная функция является рекурсивной. Из функции возвращается 1- если данный файл является путем к файлу-минотавру и 0- если это файл-тупик. Возвращаемые значения помогают регулировать рекурсию.

В функции создаётся элемент-структуру *File new_ file,* затем вызывается функция *setPath().* Таким образом получается структуру *new_file* с заполненными полями *filename* и *path*. Функцией *fopen()* файл *new_file.path* открывается на чтение**.** Считывается первая строка файла с помощью *fscanf()*, чтобы проверить, к какому типу файл отнести: файл-минотавр, файл-тупик или файл с ссылками на названия других файлов. Если это файл-минотавр, то

происходит добавление его в *Maze* функцией *push_answer(),* функцией fclose() файл закрывается и возвращается значение 1.

Если это файл-тупик, функцией *fclose()* файл закрывается и возвращается значение 0.

Если это файл со ссылками, то происходит считывание названий файлов. После считывания проверяется значение рекурсивной функции *if(findMinotaur(startDir, file, arr) == 1),* т.е. если через файл-ссылку можно каким-то образом дойти до минотавра, то и данный файл, который на него ссылается, участвует в этой цепочке.

Поэтому происходит добавление его в *Maze* функцией *push_answer(),* потом с помощью *fclose()* файл закрывается и возвращается значение 1, так как данный файл участвует в цепочке файлов.

void free_memory(char** elem, int count);

В данной функции с помощью цикла for происходит освобождение выделенной динамической памяти.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	Была создана директория *labyrinth* по представленному примеру в условии задачи.	Результат был записан в файл *result.txt* *./labyrinth/add/add/file.txt* *./labyrinth/add/mul/add/file4.txt* *./labyrinth/add/mul/file2.txt* *./labyrinth/add/mul/file3.txt*

Выводы.

В ходе выполнения лабораторной работы был изучен обход файловой системы, а также освоены функции для работы с файловой системой.

Разработана программа, которая обходит некоторую корневую директорию с вложенными папками и файлами, находит так называемый «файл-минотавр» и выводит правильную цепочку файлов(с путями), которая привела к его поимке.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Yarusova_Tatyana_lb3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/types.h>

#define answer "Minotaur"
#define end "Deadlock"

typedef struct File{
    char* filename;
    char* path;
}File;

typedef struct Maze{

    char** links;
    int count;
    int maxcount;

}Maze;

void setPath(const char* pathDir, File* path_file){
    char nextDir[300]={0};
    strcpy(nextDir,pathDir);
    strcat(nextDir,"/");
    DIR* dir=opendir(pathDir);
    if(!dir)
        return;

    struct dirent* current;
    current = readdir(dir);
    while(current){

        if((current->d_type == 4)&&(strcmp(current->d_name, ".") !=
0)&&(strcmp(current->d_name, "..") != 0)){
                int len;
                len = strlen(nextDir);
                strcat(nextDir,current->d_name);
                setPath(nextDir, path_file);
                nextDir[len] = '\0';
        }

        if((current->d_type ==  8) && (strcmp(current->d_name,
path_file->filename) == 0)){
            strcat(nextDir,current->d_name);
            path_file->path = malloc(300 * sizeof(char));
            strcpy(path_file->path,nextDir);
            closedir(dir);
            return;
        }
```

```
        current = readdir(dir);
    }

}

void push_answer(Maze* arr, char* path){
    if(arr->count == arr-> maxcount){
        arr->maxcount += 10;
        arr->links = realloc(arr, arr->maxcount * sizeof(char*));
    }

    arr->links[arr->count] = malloc((strlen(path)+2)*sizeof(char*));
    strcpy(arr->links[arr->count++], path);
}


int findMinotaur(const char* startDir, char* filename, Maze* arr){
    File new_file;
    new_file.filename = malloc(50 * sizeof(char));
    strcpy(new_file.filename, filename);

    setPath(startDir, &new_file);

    FILE* f = fopen(new_file.path,"r");
    if(!f)
        return 0;

    char s[100];
    fscanf(f,"%s",s);

    if(strcmp(s, answer) == 0){
        push_answer(arr, new_file.path);
        fclose(f);
        return 1;
    }
    else{
        if(strcmp(s, end) == 0){
            fclose(f);
            return 0;
        }
        else{

            char t[20];
            char file[100];

            fscanf(f,"%s",s);
            if(findMinotaur(startDir, s, arr) == 1){
                    push_answer(arr, new_file.path);
                    fclose(f);
                    return 1;
                }


            while(fscanf(f,"%s%s",t,s)!= EOF){
                strcpy(file,s);
                if(findMinotaur(startDir, file, arr) == 1){
                    push_answer(arr, new_file.path);
                    fclose(f);
```

```
                    return 1;
                }
            }
        }
    }
}

void free_memory(char** elem, int count){
    for(int i  = 0; i < count; i++){
        free(elem[i]);
    }
}

int main(){
    const char* startDir  = "./labyrinth";
    char* filename = "file.txt";

    Maze path;
    path.maxcount = 10;
    path.count = 0;
    path.links = malloc(path.maxcount * sizeof(char*));

    findMinotaur(startDir, filename, &path);

    FILE* res_file = fopen("result.txt", "w");
    for(int i  = path.count - 1; i >= 0; i--){
        fprintf(res_file,"%s\n",path.links[i]);
    }

    fclose(res_file);
    free_memory(path.links, path.count);
    free(path.links);
    return 0;
}
```