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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Программирование»
Тема: Обход файловой системы

Студентка гр. 1304 ____ Нго Тхи Йен

Преподаватель ____ Чайка К.В

Санкт-Петербург
2022

Цель работы.

Изучить основные методы работы с файлами и директориями. Изучить рекурсию, метод обхода дерева в глубину.

Вариант 1:

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

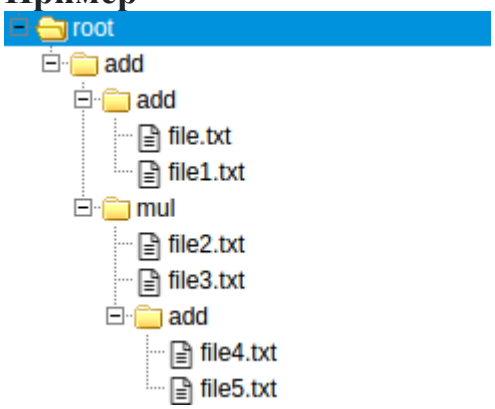
А также файл может содержать тупик:

- **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**.

Экспериментальные результаты.

Входные данные	**Выход (в result.txt)**
labyrinth add add file.txt { @include file1.txt @include file4.txt @include file5.txt} file1.txt {Deadlock} mul file2.txt {@include file3.txt} file3.txt {Minotaur} add file4.txt { @include file2.txt @include file1.txt} file5.txt {Deadlock}	./labyrinth/add/add/file.txt ./labyrinth/add/mul/add/file4.txt ./labyrinth/add/mul/file2.txt ./labyrinth/add/mul/file3.txt

Выводы.

Был изучен рекурсивный метод обхода дерева, с его помощью была обработана директория, включающая в себя файлы с ссылками на другие файлы а также вложенные директории. Результат работы программы был записан в файл.

Исходный код программы.

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

typedef struct Node{
    char* path;
    struct Node* next;
    struct Node* prev;
} Node;

Node* createNode(char* path){
     char* dir = malloc(500 * sizeof(char));
     strcpy(dir, path);
     Node* head = malloc(sizeof(Node));
     head->path = dir;
     head->next = NULL;
     head->prev = NULL;
     return head;
}

void push(Node* head, char* path){
     char* dir = malloc(500 * sizeof(char));
     strcpy(dir, path);
     Node* cur = malloc(sizeof(Node));
     cur->path = dir;
     cur->next = NULL;
     cur->prev = NULL;
     Node* n = head;
     while (n->next)
          n = n->next;
     cur->prev = n;
     n->next = cur;
}

void write(Node* head){
     Node* cur = head;
     while (cur->next)
```

```
        cur = cur->next;
    FILE* f = fopen("result.txt", "w");
    while (cur){
        fprintf(f, "%s¥n", cur->path);
        cur = cur->prev;
    }
    fclose(f);

}

void freeList(Node* head){
    Node* cur = head;
    while (cur->next){
        cur = cur->next;
        free(cur->prev);
    }
    free(cur);
}

int search(char *dir_name, char *file_name, Node** out_list){
    DIR *cur_dir = opendir(dir_name);
    struct dirent *cur_file;
    while (cur_file = readdir(cur_dir)){
        int last_end = strlen(dir_name);
        strcat(dir_name, "/");
        strcat(dir_name, cur_file->d_name);
        if (cur_file->d_type == DT_REG && !strcmp(cur_file->d_name,
file_name)){
            FILE *f = fopen(dir_name, "r");
            char content[1000];
            while (fgets(content, 1000, f)){
                if (strstr(content,"¥n") && content)
                    *strstr(content, "¥n") = '¥0';
                if (!strcmp(content, "Minotaur")){
                    *out_list = createNode(dir_name);
                    closedir(cur_dir);
                    fclose(f);
                    return 1;
                }
                if (!strcmp(content, "Deadlock")){
```

```
                        closedir(cur_dir);
                        fclose(f);
                        return 0;
                    }
                    char other_path[500];
                    strcpy(other_path, "./labyrinth");
                    int out = search(other_path, strstr(content, " ") +
1, out_list);
                    if (out){
                        push(*out_list, dir_name);
                        closedir(cur_dir);
                        fclose(f);
                        return out;
                    }
                }
            }
            if (cur_file->d_type == DT_DIR && strcmp(cur_file->d_name,
".") && strcmp(cur_file->d_name, "..")){
                int out = search(dir_name, file_name, out_list);
                if (out){
                    closedir(cur_dir);
                    return out;
                }
            }
            dir_name[last_end] = '¥0';
        }
        closedir(cur_dir);
        return 0;
}

int main(){
        Node* out_list;
        char dir_name[500];
        strcpy(dir_name, "./labyrinth");
        search(dir_name, "file.txt", &out_list);
        write(out_list);
        freeList(out_list);
        return 0;
}
```