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. 9304 _____ Каменская Е.К.

Преподаватель _____ Чайка К.В

Санкт-Петербург
2020

Цель работы.

Научиться применять основные функции языка Си для работы с файловой системой и на их основе решить задачу рекурсивного обхода файловой системы.

Задание.

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 <filename>.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.

Выполнение работы.

Функции:

- i*nt isValid(char* string)* – компилирует регулярное выражение для выделения названия файла из ссылки на него в другом файле и возвращает единицу, если переданная строка подходит под выражение и 0 в противном случае. При этом изменяет строку, указатель на которую ей передан, чтобы строка содержала только название файла.
- *void findFile(char* path, char* toFind, FILE** logPtr, char*** wayToMinotaur, int size, int cur)* – совершает рекурсивный обход директории. Ей передается название искомого файла, и, если она встречает его, вызывается функция *readFile*.
- *void readFile(char* name, char* path, FILE** logPtr, char*** wayToMinotaur, int size, int cur)* – построчно считывает информацию из переданного файла. Найдя файл-минотавр прекращает выполнение программы. В ином случае либо заносит следующий файл в динамический массив *wayToMinotaur*, при этом сдвигая итератор по массиву вперед на 1, либо сдвигает итератор cur на 1 назад. Она же записывает результат работы программы в файл *result.txt*.
- *int main()* - открывает файл *result.txt* на запись или создает его. Инициализирует массив путей *wayToMinotaur*, а также директорию поиска и первый файл в цепочке, после чего запускает *findFile*.

Выводы.

В ходе выполнения работы были изучены способы работы с файловой системой, реализована программа для поиска к файлу с искомым содержимым через рекурсию.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: solution.c

```
#include <stdio.h>
#include <dirent.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <sys/types.h>
#define HOMEPATH "./labyrinth"
#define PATTERN "@include (\\S+.txt)"

int isValid(char* string){
    regex_t regexCompiled;

    size_t maxGroups = 2;

    regmatch_t groupArray[maxGroups];

    const char* regexString = PATTERN;

    if(regcomp(&regexCompiled, regexString, REG_EXTENDED)){
        printf("Compiling error\n");
        return 0;
    }

    if(regexec(&regexCompiled, string, maxGroups, groupArray, 0)
== 0){
        memmove(string, &string[groupArray[1].rm_so], sizeof(char)
* ( groupArray[1].rm_eo - groupArray[1].rm_so ));
        string[groupArray[1].rm_eo - groupArray[1].rm_so] = '\0';
        regfree(&regexCompiled);
        return 1;
    }
    else{
        regfree(&regexCompiled);
        return 0;
    }
}

 void findFile(char* path, char* toFind, FILE** logPtr, char***
wayToMinotaur, int size, int cur);
 void readFile(char* name, char* path, FILE** logPtr, char***
wayToMinotaur, int size, int cur);

int main(){
    FILE* result = fopen("result.txt", "w");

    int size = 3000;
    int cur = 0;
    char** wayToMinotaur = malloc(sizeof(char*) * size);
    for(int i=0; i<size; i++){
        wayToMinotaur[i] = malloc(128*sizeof(char));
```

```
        }

        if(!result){
            perror ("Could not open the result file");
            return 0;
        }
        char path[9000] = HOMEPATH;
        char first[10] = "file.txt";
        findFile(path, first, &result, &wayToMinotaur, size, cur);
        return 0;
    }

    void findFile(char* path, char* toFind, FILE** logPtr, char***
wayToMinotaur, int size, int cur){
        DIR *dir = opendir(path);
        if(dir){
            char buf[9000];
            struct dirent *de = readdir(dir);
            while(de){
                if(de->d_type == DT_DIR && strcmp(de->d_name, ".")!=0
&& strcmp(de->d_name, "..")!= 0){
                    int len = strlen(path);
                    strcat(path, "/");
                    strcat(path, de->d_name);
                    findFile(path,  toFind,  logPtr,  wayToMinotaur,
size, cur);
                    path[len] = '\0';
                }
                if(de->d_type   ==   DT_REG   &&   strcmp(de->d_name,
toFind)==0){
                    strcpy(buf, path);
                    strcat(buf, "/");
                    strcat(buf, de->d_name);
                    readFile(buf, path, logPtr, wayToMinotaur, size,
cur);
                }
                de = readdir(dir);
            }
            closedir(dir);
        }
    }

    void readFile(char* name, char* path, FILE** logPtr, char***
wayToMinotaur, int size, int cur){
        FILE* f = fopen(name, "r");

        if(!f){
            perror ("Could not open the text file");
            return;
        }

        char string[100];

        while(!feof(f)){

            fgets(string, 100, f);
```

```
            if(strchr(string, '\n')){
                *(strchr(string, '\n')) = '\0';
            }

            if(cur == 0 || strcmp((*wayToMinotaur)[cur-1], name)!=0){
                strcpy((*wayToMinotaur)[cur], name);
                cur++;
            }

            if(strcmp("Minotaur", string)==0){
                for(int i=0; i<cur; i++){
                    fprintf(*logPtr, "%s\n", (*wayToMinotaur)[i]);
                }
                fclose(f);
                fclose(*logPtr);
                for(int i=0; i<size; i++){
                    free((*wayToMinotaur)[i]);
                }
                free(*wayToMinotaur);
                exit(0);
            }

            if(isValid(string)){
                strcpy(path, HOMEPATH);
                findFile(path, string, logPtr, wayToMinotaur, size,
cur);
                char c = getc(f);
                if(c==PATTERN[0]){ungetc(PATTERN[0], f);}
            }

            if(strcmp("Deadlock", string)==0 || strcmp("Deadlock ",
string)==0){
                cur = cur - 1;
                break;
            }
        }
        fclose(f);
    }
```