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

Студент гр. 3344		Анахин Е.Д.
Преподаватель		Глазунов.С.А

Санкт-Петербург
2024

Цель работы

Изучить принцип работы с рекурсивными алгоритмами и файловой системой на языке программирования Си.

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 .
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.

Выполнение работы

Выполнение работы будет расписано по шагам:

1) Написать функцию main, которая будет запускать рекурсивный обход файловой системе и искать минотавра. Затем эта функция должна записать ответ
2) Написать функцию findFileByName используется для нахождения файла в папке
3) Написать функцию findMinotaur, которая будет рекурсивно обходить систему и запоминать названия файлов, если в них содержимое начиналось с «@include» и добавлять их в список путей, если этот файл, в конечном итоге, приведет к минотавру
4) Написать функцию processString, которая бы извлекала аргументы из файла

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	file.txt: @include file1.txt file1/.txt: Minotaur	`./root/add/add/file.txt` `./root/add/add/file.txt3`	Верный ответ

Выводы

Были изучены принципы работы с рекурсивными алгоритмами и файловой системой на языке программирования Си.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: solution.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/types.h>
#include <libgen.h>
#define FILE_OPEN_ERROR "couldn't open this file or directory"
#define MEMORY_ALLOCATION_ERROR "memory allocation failed"
char* processString(char* str) {
char* newline = strchr(str, '\n');
if (newline) {
*newline = '\0';
}
char* lastSpace = strrchr(str, ' ');
if (lastSpace) {
*lastSpace = '\0';
}
if (strcmp(str, "Deadlock") == 0) {
return "Deadlock";
} else if (strcmp(str, "Minotaur") == 0) {
return "Minotaur";
} else if (strstr(str, "@include") == str) {
return str + 9;
} else {
return "Unknown string";
}
}
char* findFileByName(const char *dirPath, const char* fileName) {
DIR *dir = opendir(dirPath);
if(dir == NULL) {
perror(FILE_OPEN_ERROR);
return NULL;
}

struct dirent *de;

while ((de = readdir(dir)) != NULL) {
if(strcmp(de->d_name, ".") == 0 || strcmp(de->d_name, "..") == 0) {
continue;
}

char newDirPath[512];
snprintf(newDirPath, sizeof(newDirPath), "%s/%s", dirPath, de-
>d_name);
```

```
if (strcmp(basename(newDirPath), fileName) == 0) {
char *result = malloc(strlen(newDirPath) + 1);
if (result == NULL) {
perror(MEMORY_ALLOCATION_ERROR);
exit(EXIT_FAILURE);
}
strcpy(result, newDirPath);
closedir(dir);
return result;
}

if(de->d_type == DT_DIR) {
char *found = findFileByName(newDirPath, fileName);
if (found != NULL) {
closedir(dir);
return found;
}
}
}

closedir(dir);
return NULL;
}
char** readFile(char* filename) {
char** pathNames = malloc(1 * sizeof(char*));
int len = 0;
FILE* file = fopen(filename, "r");
if (file == NULL) {
printf(FILE_OPEN_ERROR);
return NULL;
}
char* stringInFile = malloc(200);
while (fgets(stringInFile, 200, file)) {
pathNames[len] = malloc(strlen(stringInFile) + 1);
if (pathNames[len] == NULL) {
perror(MEMORY_ALLOCATION_ERROR);
exit(EXIT_FAILURE);
}
strcpy(pathNames[len++], stringInFile);
pathNames = realloc(pathNames, (len + 1) * sizeof(char*));
if (pathNames == NULL) {
perror(MEMORY_ALLOCATION_ERROR);
exit(EXIT_FAILURE);
}
}
free(stringInFile);
pathNames[len] = NULL;
return pathNames;
}
```

```
void findMinotaur(char* filename, int* wasFound, char*** visitedFiles,
int* visitedCount) {
if (*wasFound) {
return;
}

char* foundFile = findFileByName("./labyrinth", filename);
if (foundFile == NULL) {
printf(FILE_OPEN_ERROR);
return;
}
*visitedFiles = realloc(*visitedFiles, (*visitedCount + 1) *
sizeof(char*));

if (*visitedFiles == NULL) {
perror(MEMORY_ALLOCATION_ERROR);
exit(EXIT_FAILURE);
}

(*visitedFiles)[*visitedCount] = foundFile;
(*visitedCount)++;
char** foundFiles = readFile(foundFile);
char* argToFind = malloc(1000);
for (int i = 0; foundFiles[i]; i++) {
strcpy(argToFind, processString(foundFiles[i]));
if (strcmp(argToFind, "Minotaur") == 0) {
*wasFound = 1;
free(argToFind);
for (int j = 0; foundFiles[j]; j++) {
free(foundFiles[j]);
}
free(foundFiles);
return;
}
if (strcmp(argToFind, "Deadlock") == 0) {
free(argToFind);
for (int j = 0; foundFiles[j]; j++) {
free(foundFiles[j]);
}
(*visitedCount)--;
free((*visitedFiles)[*visitedCount]);
*visitedFiles = realloc(*visitedFiles, (*visitedCount) *
sizeof(char*));
if (*visitedFiles == NULL && *visitedCount > 0) {
perror(MEMORY_ALLOCATION_ERROR);
exit(EXIT_FAILURE);
}
free(foundFiles);
return;
}
```

```
findMinotaur(argToFind, wasFound, visitedFiles, visitedCount);
if (*wasFound) {
free(argToFind);
for (int j = 0; foundFiles[j]; j++) {
free(foundFiles[j]);
}
free(foundFiles);
return;
}
}
free(argToFind);
for (int j = 0; foundFiles[j]; j++) {
free(foundFiles[j]);
}
free(foundFiles);
(*visitedCount)--;
free((*visitedFiles)[*visitedCount]);
*visitedFiles = realloc(*visitedFiles, (*visitedCount) *
sizeof(char*));
if (*visitedFiles == NULL && *visitedCount > 0) {
perror(MEMORY_ALLOCATION_ERROR);
exit(EXIT_FAILURE);
}
}
int main() {

int wasFound = 0;
char** visitedFiles = NULL;
int visitedCount = 0;
findMinotaur("file.txt", &wasFound, &visitedFiles, &visitedCount);
FILE *file = fopen("result.txt", "w");
if (file == NULL) {
printf(FILE_OPEN_ERROR);
}
for (int i = 0; i < visitedCount; i++) {
if (i < visitedCount - 1) {
fprintf(file, "%s\n", visitedFiles[i]);
} else {
fprintf(file, "%s", visitedFiles[i]);
}
free(visitedFiles[i]);
}
free(visitedFiles);
return 0;
}
```