**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

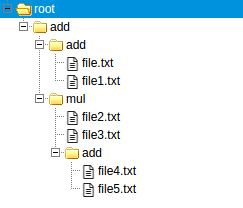
А также файл может содержать тупик:

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.

1. 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.

1. Функция char\* getFilename(char\* buf) из строки, содержащейся в файле достает название файла, который ищет функция find\_file.
2. 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 |
|  | ./labyrinth | ./labyrinth/add/add/file.txt  ./labyrinth/add/mul/add/file4.txt  ./labyrinth/add/mul/file2.txt  ./labyrinth/add/mul/file3.txt | Тест с сайта e.moevm |
|  | ./test | ./test/file.txt  ./test/pop/file3.txt  ./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;

}

//Поиск файла

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;

}