**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. 3342 |  | Русанов А.В. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Ознакомление с рекурсией, а также с её применением для обхода файловой системы с помощью языка программирования С.

## Задание

Вариант 3.

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

В каждом текстовом файле хранится одна строка, начинающаяся с числа вида:

<число><пробел><латинские буквы, цифры, знаки препинания> ("124 string example!")

Требуется написать программу, которая, будучи запущенной в корневой директории, выведет строки из файлов всех поддиректорий в порядке возрастания числа, с которого строки начинаются

## Выполнение работы

Функция comparator - функция сравнения двух строк, используется для сортировки массива строк в порядке возрастания чисел, содержащихся в строках.

Функция memory\_allocation\_check - функция, которая проверяет, выделена ли память успешно. Если указатель равен NULL, то выводится сообщение об ошибке и программа завершается.

Функция file\_is\_valid - функция проверяет, является ли файл допустимым для обработки. Возвращает 1, если файл имеет расширение ".txt" и не имеет названия "result.txt", иначе возвращает 0.

Функция dir\_is\_valid - функция проверяет, является ли директория допустимой для обработки. Возвращает 1, если директория не является текущей директорией (".") или родительской директорией (".."), иначе возвращает 0.

Функция get\_full\_path - функция для получения полного пути к файлу или директории, объединяя две строки пути с помощью символа "/".

Функция read\_file\_data - функция для чтения данных из файла. Функция открывает файл для чтения, считывает его содержимое и сохраняет каждую строку в массиве строк lines, увеличивая его размер при необходимости.

Функция read\_lines - рекурсивная функция для чтения содержимого директории. Она открывает директорию, считывает содержимое и выполняет определенные действия для каждого файла и поддиректории внутри нее.

Функция print\_result - функция для печати результатов в файл. Она принимает открытый файловый указатель file, массив строк lines и количество строк count\_lines, и записывает содержимое массива строк в файл.

Функция free\_memory - функция для освобождения памяти. Она освобождает память, выделенную для массива строк lines и для строки с полным путем new\_dir.

Функция main - основная функция программы. Сначала она получает текущую директорию, затем создает переменные для хранения количества строк count\_lines и массива строк lines. Затем она вызывает функцию read\_lines для чтения данных из файлов в текущей директории (и поддиректориях), сортирует массив строк с помощью функции qsort, создает новый файл с именем "result.txt" и выводит результаты в него с помощью функции print\_result. После этого память освобождается с помощью функции free\_memory.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1. |  | 1 Small text  2 Simple text  3 Wow? Text?  4 Where am I?  5 So much files! | Верное содержимое файла |

## Выводы

Было проведено ознакомление с рекурсией. Разработана программа на языке программирования С с использованием библиотеки dirent.h для реализации обхода файловой системы.

# Приложение А Исходный код программы

Название файла: main.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

#include <regex.h>

#include <unistd.h>

#define MAX\_PATH\_SIZE 512

int comparator(const void\* a, const void\* b)

{

char\*\* line\_a = (char\*\*)a;

char\*\* line\_b = (char\*\*)b;

long number\_a = atol(\*line\_a);

long number\_b = atol(\*line\_b);

if (number\_a < number\_b)

return -1;

if (number\_a > number\_b)

return 1;

return 0;

}

void memory\_allocation\_check(void\* pointer)

{

if(pointer == NULL)

{

printf("Memory allocation error!\n");

exit(1);

}

}

int file\_is\_valid(char \*filename)

{

return (strstr(filename, ".txt") && !strstr(filename, "result.txt"));

}

int dir\_is\_valid(char \*dir\_name)

{

return (strcmp(dir\_name, ".") != 0 && strcmp(dir\_name, "..") != 0);

}

char \*get\_full\_path(const char \*path1, const char \*path2)

{

int res\_path\_len = strlen(path1) + strlen(path2) + 2;

char \*res\_path = malloc(res\_path\_len \* sizeof(char));

memory\_allocation\_check(res\_path);

sprintf(res\_path, "%s/%s", path1, path2);

return res\_path;

}

void read\_file\_data(char \*filepath, char \*\*\*lines, int \*count\_lines)

{

FILE \*f = fopen(filepath, "r");

if (f)

{

char \*line = calloc(100, sizeof(char));

memory\_allocation\_check(line);

int len\_line = 0;

char sym;

while ((sym = fgetc(f)) != EOF)

{

if (sym == '\n')

{

continue;

}

line[len\_line++] = sym;

if(len\_line >= sizeof(line) / sizeof(char))

{

line = realloc(line, sizeof(char) \* (len\_line + 20));

memory\_allocation\_check(line);

}

}

line[len\_line] = '\0';

(\*lines)[(\*count\_lines)++] = line;

\*lines = (char \*\*)realloc(\*lines, sizeof(char \*) \* (\*count\_lines + 1));

memory\_allocation\_check(lines);

}

fclose(f);

}

void read\_lines(const char \*dir\_name, char \*\*\*lines, int \*count\_lines)

{

DIR \*dir = opendir(dir\_name);

if (dir)

{

struct dirent \*de = readdir(dir);

while (de)

{

if (de->d\_type == DT\_REG && file\_is\_valid(de->d\_name))

{

char \*new\_dir = get\_full\_path(dir\_name, de->d\_name);

read\_file\_data(new\_dir, lines, count\_lines);

free(new\_dir);

}

else if (de->d\_type == DT\_DIR && dir\_is\_valid(de->d\_name))

{

char \*new\_dir = get\_full\_path(dir\_name, de->d\_name);

read\_lines(new\_dir, lines, count\_lines);

free(new\_dir);

}

de = readdir(dir);

}

closedir(dir);

}

else

printf("Failed to open %s directory\n", dir\_name);

}

void print\_result(FILE\* file, char\*\* lines, int\* count\_lines)

{

fprintf(file, "%s", lines[0]);

for (int i = 1; i < (\*count\_lines); i++)

{

fprintf(file, "\n%s", lines[i]);

}

}

void free\_memory(char\* new\_dir, char\*\* lines, int\* count\_lines)

{

free(new\_dir);

for (int i = 0; i < (\*count\_lines); i++)

{

free(lines[i]);

}

free(lines);

}

int main()

{

char current\_dir[MAX\_PATH\_SIZE];

if (!getcwd(current\_dir, sizeof(current\_dir)))

{

perror("getcwd");

exit(1);

}

int tmp = 0;

int \*count\_lines = &tmp;

char \*\*lines = (char \*\*)malloc(sizeof(char \*));

memory\_allocation\_check(lines);

read\_lines(current\_dir, &lines, count\_lines);

qsort(lines, (\*count\_lines), sizeof(char\*), comparator);

char\* new\_dir = get\_full\_path(current\_dir, "/result.txt");

FILE \*file = fopen(new\_dir, "w+");

if (!file){

printf("Error: cannot open result.txt\n");

exit(1);

}

print\_result(file, lines, count\_lines);

free\_memory(new\_dir, lines, count\_lines);

fclose(file);

return 0;

}