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. 1304 _____ Шаврин А.П

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2022

Цель работы.

Научиться работать с файловой системой с помощью языка программирования Си, а также обходить дерево файлов при помощи рекурсии.

Задание.

Вариант – 3

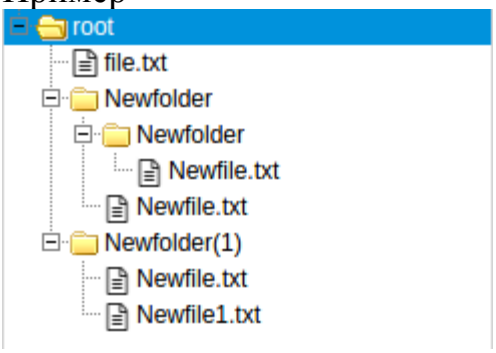
Дана некоторая корневая директория, в которой может находиться некоторое количество папок, в том числе вложенных. В этих папках хранятся некоторые текстовые файлы, имеющие имя вида *.txt*

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

Пример



root/file.txt: 4 Where am I?
root/Newfolder/Newfile.txt: 2 Simple text
root/Newfolder/Newfolder/Newfile.txt: 5 So much files!
root/Newfolder(1)/Newfile.txt: 3 Wow? Text?
root/Newfolder(1)/Newfile1.txt: 1 Small text

Решение:

1 Small text
2 Simple text
3 Wow? Text?
4 Where am I?
5 So much files!

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***.

Основные теоретические положения.

Рекурсия

Работа с деревом файловой системы

Выполнение работы.

Для решения данной задачи было решено использовать рекурсивный алгоритм обхода дерева файловой системы.

Изначально было объявлено несколько констант:

- *ROOT_DIRECTORY*
- *OUTPUT_FILE_NAME*
- *SEARCH_FILE_TYPE*
- *PATH_SIZE* (количество байт выделяемое под полный путь до файла или катологa)
- *MAX_COUNT_FILES*
- *CONTENT_LINE_LENGTH* (количество байт выделяемое под запись строки из файла)

Затем создана структура *FileContent,* хранящая в себе 2 поля *int number u char *string*. После была описана функция *FileContent CreateFileContent(char *content)*, возвращающая на основе считанных данных описанную выше структуру. Также была описана функция чтения файла *int ReadFile(FileContent *FilesArray, char *file_path, int count_elem)*, которая возвращает 1, если файл удалось прочитать и 0 в противном случае, и также функция void FreeFileContent(FileContent *elem), которая освобождает память выделенную под структуру и ее поля. После была реализована основная рекурсивная функция обхода дерева файловой системы *int DirectoryRecursion(char *DirectoryPath, FileContent *FilesArray)*, которая возвращает количество прочитанных файлов, основываясь на значениях возвращаемых функцией чтения файла. В самую последнюю очередь были реализованы функции *int CompareFileContent(const void *a, const void *b)*, для сравнения контента считанного из файлов и дальнейшей сортировки этого контента, и *void PrintFilesArrayToFile(FileContent *FilesArray, int count_elem, char* output_file_name)*, для записи результата в итоговый файл.

В функции main происходит создание массива структур, содержащих контент файла, затем обход файловой системы, сортировка массива и последующая его запись в файл.

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.	test └── 1 ├── 1.txt (1 asdfghj) ├── 2 │ ├── 21.txt (21 qwertyu) │ └── 2.txt (2 qwert) ├── 3 │ └── 3.txt (3 zxcvb3) ├── 4 └── 4.txt (4 ghjkl) ├── 5 └──5.txt(5 uioppoiu)	1 asdfghj 2 qwert 3 zxcvb3 4 ghjkl 5 uioppoiu 21 qwertyu	Результат корректен

Выводы.

В ходе выполнения данной лабораторной работы была изучена работа с файловой системой языка программирования Си, а также была создана программа, выполняющая обход дерева файловой системы.

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>


#define ROOT_DIRECTORY "."
#define OUTPUT_FILE_NAME "result.txt"
#define SEARCHING_FILE_TYPE ".txt"
#define PATH_SIZE 1024
#define MAX_COUNT_FILES 16384
#define CONTENT_LINE_LENGTH 256


typedef struct FileContent{
	long int number;
	char *string;
} FileContent;


int CompareFileContent(const void *a, const void *b){
	//type casting
	FileContent *f = (FileContent *)a;
	FileContent *s = (FileContent *)b;

	//compare
	if (f -> number > s -> number)
```

```
        return 1;
    if (f -> number < s -> number)
        return -1;
    else
        return 0;
}
FileContent CreateFileContent(char *content){
    //create element
    FileContent elem;
    elem.number = 0;
    elem.string = NULL;

    //get space position
    char *posSpace = strstr(content, " ");

    //check if the string is correct
    if (!posSpace)
        return elem;

    //initialize element
    elem.number = atoi(content);
    elem.string = (char*)malloc(strlen(posSpace)*sizeof(char));
    strcpy(elem.string, posSpace + 1);

    return elem;
}


int ReadFile(FileContent *FilesArray, char *file_path, int count_elem){
    FILE *file = fopen(file_path, "r");
```

```
        if (file != NULL){

            //read information from a file
            char *content = malloc(CONTENT_LINE_LENGTH*sizeof(char));
            fgets(content, CONTENT_LINE_LENGTH, file);

            //create file content and add him into FilesArray
            FilesArray[count_elem] = CreateFileContent(content);

            //cleaning
            free(content);

            fclose(file);
            return 1;
        }
        return 0;
}
int DirectoryRecursion(char *DirectoryPath, FileContent *FilesArray){

        //error checking
        if (DirectoryPath == NULL || FilesArray == NULL)
            return 0;

        DIR *current_directory = opendir(DirectoryPath);
        //error checking
        if (current_directory == NULL)
            return 0;

        //count elem in the files array
        int count_elem = 0;
```

```
//get first directory element
struct dirent *dp = readdir(current_directory);

//itarate over all elem of a directory
while (dp){

    //checking for start directory or exit
    if (dp -> d_type == DT_DIR && (strcmp(dp -> d_name, ".") == 0 ||
strcmp(dp -> d_name, "..") == 0)){
        dp = readdir(current_directory);
        continue;
    }

    //checking for output file
        if (dp -> d_type == DT_REG && (!strcmp(dp -> d_name,
OUTPUT_FILE_NAME) || strstr(dp->d_name, SEARCHING_FILE_TYPE) ==
NULL)){
        dp = readdir(current_directory);
        continue;
    }

    //create path to the file or directory
    char path[PATH_SIZE];
    strcpy(path, DirectoryPath);
    strcat(path, "/");
    strcat(path, dp -> d_name);

    //file type checking
            if (dp -> d_type == DT_REG && strstr(dp->d_name,
SEARCHING_FILE_TYPE) != NULL){
        count_elem += ReadFile(FilesArray, path, count_elem);
```

```
            }else{
                if (dp -> d_type == DT_DIR){
                    count_elem += DirectoryRecursion(path, FilesArray +
count_elem);
                }
            }
        /get new directory element
            dp = readdir(current_directory);
        }

        closedir(current_directory);
        return count_elem;
    }


    void FreeFileContent(FileContent *elem){
        if (elem != NULL){
            free(elem -> string);
            elem -> string = NULL;
            elem -> number = 0;
        }
    }


    void PrintFilesArrayToFile(FileContent *FilesArray, int count_elem, char*
output_file_name){
        FILE* output_file = fopen(output_file_name, "w");
        //error checking
        if (output_file == NULL){
            printf("I can't open outpute file\n");
            return;
```

```
        }

        //error checkin
        if (FilesArray == NULL)
            return;

        for (int i = 0; i < count_elem; i++){
            if (FilesArray[i].string == NULL){continue;}
                    fprintf(output_file,  "%ld  %s\n",  FilesArray[i].number,
FilesArray[i].string);
            FreeFileContent(FilesArray + i);
        }
        fclose(output_file);
    }
    int main(){
        //create files array
                                        FileContent     *FilesArray     =
malloc(MAX_COUNT_FILES*sizeof(FileContent));

        //get info and count files in array
        int count_elem = DirectoryRecursion(ROOT_DIRECTORY, FilesArray);

        //sort array
        qsort(FilesArray, count_elem, sizeof(FileContent), CompareFileContent);

        //print result
        PrintFilesArrayToFile(FilesArray, count_elem, OUTPUT_FILE_NAME);

        //free array
        free(FilesArray);
        FilesArray = NULL;
```

```
    return 0;

}
```