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: Строки. Рекурсия, циклы, обход дерева

Студент гр. 1304 _____ Спасов Д.В.

Преподаватель _____ Чайка К. В.

Санкт-Петербург

2022

Цель работы.

Научиться работать с файлами и директориями при помощи языка Си. Изучить рекурсивный метод обхода дерева в глубину.

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

*Файл с решением должен называться **solution.c**. Результат работы программы должен быть записан в файл **result.txt**.*

Выполнение работы.

Структуры: Перечень структур представлен в табл. 1

Таблица 1 – Структуры программы

Имя структуры	Поле структуры	Комментарий	Комментарии
FileInfo	Number	Число для сравнения строк.	Информация о файле, с которого считали строку для дальнейшей обработки.
	Str	Строка для отабражения.	

Функции: Перечень функций представлен в табл. 2

Таблица 2 – Функции программы

Имя функции	Возвращаемое значение	Аргументы	Комментарии
ParseString	Инициализированная структура информации о файле.	*char* str* — строка, считанная из файла.	Создает и инициализирует структуру информации о файле. Строка имеет формат <число> <текст>.
CompareFileInfo	Число < 0, если первая структура меньше второй структуры. Число =0, если структуры равны. Число >0, если первая структура больше второй.	*void* A*- Указатель на первую структуру для сранения. *void* B*- Указатель на вторую структуру для сравнения.	Используется в сортровке. Сравнивает две структуры информации о файле по полю *Number*.
PrintFileInfoToFile	-	*FileInfo* FileInfo* — Информация о файле для печати. *FILE* File* — Файл, куда будет производиться запись.	Записывает информацию о файле в файл.

FreeFileInfo	-	*FileInfo* FileInfo* - Информация о файле, которую нужно деинициализировать.	Очищает структуру информации о файле.
IterateDirectory	Количество файлов, которые были сохранены для дальнейшей обработки.	*char* DirPath — Путьдо корневой папки для итерирования по дочерним файлам и папкам.* *FileInfo* FilesInfo*- Массив, куда будет сохраняться информация о посещенных файлах.	Рекурсивно проходится по всем дочерним файлам и папкам, собирая необходимую информацию из файлов для дальнейшей обработки.

Тестирование.

Результаты тестирования представлены в табл. 3.

Таблица 3 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	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	1 Small text 2 Simple text 3 Wow? Text? 4 Where am I? 5 So much files!	Ответ верный.

Выводы.

Был изучен принцип работы с файлами и директориями, применен метод рекурсивного обхода дерева в глубину. По итогу написана программа, которая перебирает все файлы в текущей директории и всех поддиректориях, собирает информацию из файлов, обрабатывает ее и записывает в файл.

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 OUTPUT_FILE_NAME "result.txt"
#define ROOT_DIRECTORY "."

#define PATH_BUFFER_SIZE 1024
#define MAX_FILES_COUNT 16384


struct FileInfo
{
    long int Number;
    char* Str;
};

struct FileInfo ParseString(char* str)
{
    struct FileInfo Result;
    Result.Number = 0;
    Result.Str = NULL;

    char* FirstSpace = strstr(str, " ");
    if( FirstSpace == NULL) return Result;

    Result.Number = atoi(str);
   Result.Str = (char*)malloc(sizeof(char)*(strlen(FirstSpace + 1)+1));
    strcpy(Result.Str, FirstSpace + 1);

    return Result;
}
```

```
int CompareFileInfo(const void* A, const void* B)
{
     if( ((struct FileInfo*)A)->Number > ((struct FileInfo*)B)->Number )
      return 1;
     if( ((struct FileInfo*)A)->Number < ((struct FileInfo*)B)->Number )
      return -1;
    return 0;
}


void PrintFileInfoToFile(const struct FileInfo* FileInfo, FILE* File)
{
    if( FileInfo == NULL || File == NULL ) return;

    fprintf(File, "%ld %s\n", FileInfo->Number, FileInfo->Str);
}

void FreeFileInfo(struct FileInfo* FileInfo)
{
    if( FileInfo == NULL ) return;

    free(FileInfo->Str);
    FileInfo->Str = NULL;
    FileInfo->Number = 0;
}




int IterateDirectory(char* DirPath, struct FileInfo* FilesInfo)
{
    if( DirPath == NULL || FilesInfo == NULL ) return 0;

    DIR* CurrentDirectory = opendir(DirPath);
    if( CurrentDirectory == NULL ) return 0;


    int ElemsCount = 0;

    struct dirent* LSubDir = readdir(CurrentDirectory);
    while( LSubDir )
    {
      if( LSubDir->d_name[0] == '.')
      { LSubDir = readdir(CurrentDirectory); continue; }
      if( strcmp(LSubDir->d_name, OUTPUT_FILE_NAME) == 0 )
      { LSubDir = readdir(CurrentDirectory); continue; }

        char LFilePath[PATH_BUFFER_SIZE];
        #ifdef _WIN32
            strcpy(LFilePath, CurrentDirectory->dd_name);
            LFilePath[strlen(LFilePath) - 1] = '\0';
            strcat(LFilePath, "\\");
            strcat(LFilePath, LSubDir->d_name);
        #else
```

```
            #ifdef___linux_____________________________ strcpy(LFilePath,
                DirPath);strcat(LFilePath,
                "/");
                strcat(LFilePath, LSubDir->d_name);
            #endif //___linux_____________________
        #endif // _WIN32

        if(strstr(LSubDir->d_name, ".txt") != NULL)
        {
            FILE* LFile = fopen(LFilePath, "r");
            if(LFile == NULL)
           { LSubDir = readdir(CurrentDirectory); continue; }

            char LStr[256];
            fgets(LStr, 256, LFile);
            FilesInfo[ElemsCount] = ParseString(LStr);
            fclose(LFile);

            ++ElemsCount;
        }
        else
        {
           ElemsCount+=IterateDirectory(LFilePath,FilesInfo+ElemsCount);
        }

        LSubDir = readdir(CurrentDirectory);
    }

    closedir(CurrentDirectory);
    return ElemsCount;
}


int main()
{
    struct FileInfo FilesInfo[MAX_FILES_COUNT];

    int ElemsCount = IterateDirectory(ROOT_DIRECTORY, FilesInfo);
qsort(FilesInfo, ElemsCount, sizeof(struct FileInfo), CompareFileInfo);


    FILE* OutputFile = fopen(OUTPUT_FILE_NAME, "w");
    if(OutputFile == NULL)
    {
        puts("Can't open output file!");
        return 0;
    }

    for(int i = 0; i < ElemsCount; ++i)
    {
        PrintFileInfoToFile(&FilesInfo[i], OutputFile);
        FreeFileInfo(&FilesInfo[i]);
    }
    fclose(OutputFile);
    return 0;
}
```