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. 3341 _____ Пчелкин Н.И.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Цель работы заключается в разработке программы на языке программирования, которая осуществляет рекурсивный обход иерархии папок и файлов в заданной структуре, анализирует содержимое текстовых файлов, выполняет математические операции в соответствии с правилами задания и выводит на экран итоговый результат вычислений, основанный на содержимом файлов и вложенных папок.

Задание

Вариант 2

Задана иерархия папок и файлов по следующим правилам:

название папок может быть только "add" или "mul"

В папках могут находиться другие вложенные папки и/или текстовые файлы

Текстовые файлы имеют произвольное имя с расширением .txt

Содержимое текстовых файлов представляет собой строку, в которой через пробел записано некоторое количество целых чисел

Требуется написать программу, которая, запускается в корневой директории, содержащей одну папку с именем "add" или "mul" и вычисляет и выводит на экран результат выражения состоящего из чисел в поддиректориях по следующим правилам:

Если в папке находится один или несколько текстовых файлов, то математическая операция определяемая названием папки (add = сложение, mul = умножение) применяется ко всем числам всех файлов в этой папке

Если в папке находится еще одна или несколько папок, то сначала вычисляются значения выражений, определяемые ими, а после используются уже эти значения

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 tmp.

Основные теоретические положения

Основные теоретические положения для работы с файловой иерархией в C и использования рекурсии:

1. Работа с файловой иерархией: файловая иерархия представляет собой структуру, в которой файлы и директории организованы в виде дерева. Взаимодействие с файловой иерархией включает операции чтения, записи, создания, удаления файлов и директорий.

2. Работа с файлами и директориями: для работы с файловой иерархией в языке программирования C используются функции стандартной библиотеки языка, такие как fopen(), fclose(), fread(), fwrite(), opendir(), readdir(), closedir() и другие. Эти функции позволяют осуществлять доступ к файлам и директориям, выполнять чтение и запись данных.

3. Рекурсия: рекурсия в программировании — это прием, при котором функция вызывает саму себя. При работе с файловой иерархией рекурсия позволяет обходить все уровни директорий и файлов вложенных структур. Это особенно полезно при неопределенном количестве уровней вложенности или при необходимости выполнить однотипную операцию на каждом уровне.

4. Рекурсивное обход директорий: в рамках обработки файловой иерархии рекурсия часто используется для обхода всех элементов директории, включая поддиректории. Это позволяет пройти по всем уровням вложенности и обработать каждый файл или директорию в структуре.

5. Базовый и рекурсивный случаи: в рекурсивной функции для обхода директорий важно определить базовый случай, при котором рекурсия завершится, и рекурсивный случай, в котором функция вызывает саму себя для обработки следующего уровня директории.

6. Управление памятью: при работе с файловой иерархией и использовании рекурсии важно правильно управлять памятью. Необходимо освобождать ресурсы, выделенные для открытия файлов и директорий, чтобы избежать утечек памяти и повысить производительность программы.

Выполнение работы

Были реализованы следующие структуры:

Структура Line, означающая содержащуюся в файле строку:

long long num – число в начале файла;

char* text – последующий текст;

Были реализованы следующие функции:

char* pathcat – принимает на вход текущий путь и путь к файлу или директории. Создает путь до указанного файла или директории.

void line_finder – функция обработки файла. Заполняет соответствующие поля структуры Line, исходя из данных файла.

void file_searcher – рекурсивная функция обхождения всей директории.

void output – функция, выводящая результат программы в файл.

int cmp – функция-компаратор. Сравнивает две структуры Line по полю num.

Работа программы заключается в прохождении всей директории, нахождению необходимых файлов и считывания с них информации в массив структур Line**. Затем этот массив сортируется по полю num и отсортированный массив выводится 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.	HeLlO	./tmp/asdfgh/mkoipu/H.txt ./tmp/qwerty/e.txt ./tmp/qwerty/qwert/L.txt ./tmp/asdfgh/l.txt ./tmp/asdfgh/O.txt7	Программа обрабатывает директорию и выбирает файлы из одной буквы, чьи названия образуют заданную последовательность
2.	abc	./tmp/sub1/a.txt ./tmp/sub1/b.txt ./tmp/sub2/c.txt	Ищется заданная последовательность, притом игнорируются файлы с названием из нескольких букв
3.	aBc	./tmp/sub1/a.txt ./tmp/sub2/B.txt ./tmp/sub2/c.txt	Ищется заданная последовательность, притом игнорируются файлы с названием из нескольких букв и файлы с нужными буквами неподходящего регистра

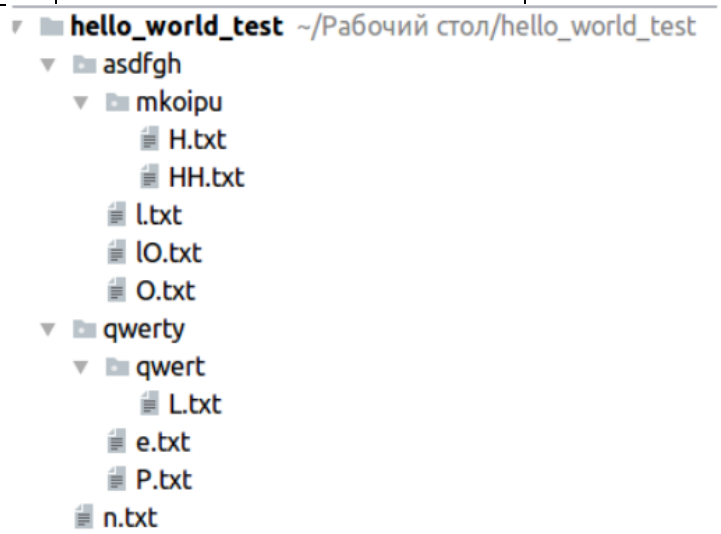

Рисунок 1- Файловое дерево для тестирования №1

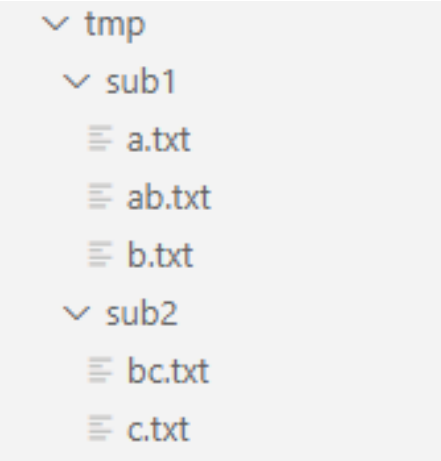

Рисунок 2 – Файловое дерево для тестирования №2

Рисунок 3 – Файловое дерево для тестирования №3

Выводы

В ходе выполнения данной работы были приобретены навыки эффективного использования рекурсивных методов для обхода сложных структур данных, а также работы с файловой системой, анализа содержимого текстовых файлов и выполнения математических операций в соответствии с заданными правилами. Разработка программы, способной автоматически обрабатывать информацию из различных файлов и директорий, позволила улучшить навыки программирования и решения сложных задач.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: solution.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dirent.h>

#define MAX_STRING 1000
#define BLOCK 1000
#define RESET_DIR ".."
#define PREV_DIR "."
#define CUR_DIR "./"
#define TEXT_FOR ".txt"


typedef struct Line{
    long long num;
    char* text;
} Line;

char* pathcat(char* cur_path, char* new_dir){
    char* new_path = malloc((strlen(cur_path) + strlen(new_dir) + 2)
* sizeof(char));
    sprintf(new_path, "%s/%s", cur_path, new_dir);
    return new_path;
}

void line_finder(char* fpath, Line** line_list, long long*
number_of_lines){
    FILE* file = fopen(fpath, "r");

    if(file){
        fscanf(file, "%lld ", &((*line_list)[*number_of_lines].num));

        int line_size = 0;
        int capacity = 1;
        char *new_line = (char *)calloc(1, BLOCK * sizeof(char));

        char new_char;
        while((new_char = fgetc(file)) != EOF){
            new_line[line_size] = new_char;
            line_size++;
            if(line_size - 1 >= capacity * BLOCK){
                capacity *= 2;
                new_line = (char *)realloc(new_line, capacity * BLOCK
* sizeof(char));
            }
        }

        (*line_list)[*number_of_lines].text = new_line;
        (*number_of_lines)++;
    }
```

```
        fclose(file);
    }

    void file_searcher(char* cur_path, Line** line_list, long long*
number_of_lines, int* capacity){
        DIR* cur_dir = opendir(cur_path);

        if(cur_dir){
            struct dirent* dir = readdir(cur_dir);
            while(dir){
                char* new_path = pathcat(cur_path, dir->d_name);

                if(dir->d_type   ==   DT_REG   &&   strstr(dir->d_name,
TEXT_FOR) != NULL){
                    if((*capacity)*BLOCK < (*number_of_lines) + 1){
                        (*capacity)++;
                        *line_list     =     (Line*)realloc(*line_list,
(*capacity) * BLOCK * sizeof(Line));
                    }
                    line_finder(new_path, line_list, number_of_lines);
                } else if(dir->d_type == DT_DIR && strcmp(dir->d_name,
PREV_DIR) != 0 && strcmp(dir->d_name, RESET_DIR) != 0){
                    file_searcher(new_path, line_list, number_of_lines,
capacity);
                }
                dir = readdir(cur_dir);
            }
            closedir(cur_dir);
        }
    }

    void output(Line** line_list, long long number_of_lines, char*
file_name){
        FILE *file = fopen(file_name, "w");

        if(file){
            for(long long i = 0; i < number_of_lines; i++){
                fprintf(file,    "%lld    %s",    (*line_list)[i].num,
(*line_list)[i].text);
                if(i < number_of_lines - 1)
                    fprintf(file, "\n");
            }
        }
        fclose(file);
    }

    int cmp(const void* a, const void* b){
        return ((Line*)a)->num > ((Line*)b)->num;
    }

    int main(){
        Line* line_list = (Line*)calloc(BLOCK, sizeof(Line));
        long long number_of_lines = 0;
        int capacity = 1;

        file_searcher(CUR_DIR, &line_list, &number_of_lines, &capacity);
        qsort(line_list, number_of_lines, sizeof(Line), cmp);
        output(&line_list, number_of_lines, "./result.txt");
```

```
    return 0;
}
```