**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 _____ Самокрутов А.Р.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

**Цель работы**

Целью работы является освоение работы с рекурсивными функциями и файловой системой, а также ее рекурсивным обходом.

Для достижения поставленной цели требуется решить следующие задачи:

1. Ознакомиться с понятием рекурсии;
2. Освоить написание рекурсивных функций в языке Си;
3. Изучить работу с файловой системой в языке Си;
4. Написать программу для рекурсивного обхода всех файлов в папке в том числе во вложенных папках.

**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

Пример
(Программа в момент запуска находится в директории root)

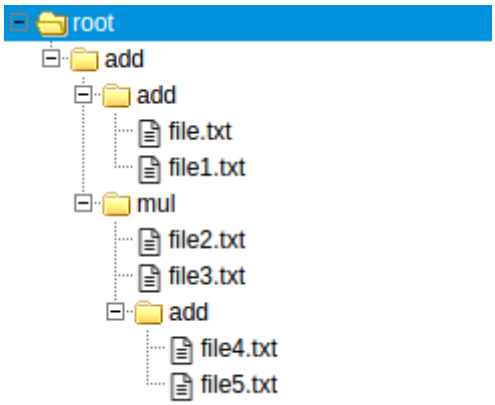


Рисунок 1 – Пример

file.txt: 1
file1.txt: 1
file2.txt: 2 2
file3.txt: 7
file4.txt: 1 2 3
file5.txt: 3 -1

Решение:

226
Выражение в данном случае имеет вид: (((1+1))+((1+2+3+3+-1)*7*2*2))

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.

**Выполнение работы**

Сначала с помощью директивы include подключаются библиотеки *sys/types.h*, *dirent.h*, *stdio.h*, *stdlib.h*, *string.h*.

Далее описаны несколько функций для работы с файлами.

Функция *char *pathcat(const char *path1, const char *path2)* принимает на вход две строки — путь к директории и имя файла в этой директории и возвращает динамически выделенную строку, в которой хранится адрес файла, образованное по принципу *<путь к директории>/<имя файла>*.

Функция *long long int add(const char *dir_name)* принимает имя директории и возвращает сумму всех элементов в ней по правилу, описанному в задании. Объявляется целое число *sum* — сумма, инициализированная нулём. Далее с помощью функции *opendir()* по переданному имени открывается директория, после чего с помощью цикла *while()* читается её содержимое. С помощью функций *is_child_dir()* и *is_txt_file()* определяется, является ли элемент в директории дочерней директорией или текстовым файлом, после чего вызывается соответственно функция *add_dir()* или *add_file()*, которые считают сумму своего содержимого и записывают её в переменную *sum*. Функция *long long int mul(const char *dir_name)* аналогична и отличается лишь тем, что считает не сумму, а произведение с помощью функции *mul_dir()* или *mul_file()*.

Функция *int is_child_dir(const struct dirent *de)* проверяет, что элемент *de* является директорией (*de->d_type == DT_DIR*) и при этом не является родительской или текущей директорией. Возвращает *0* в случае невыполнения условий, не *0* иначе. Функция *int is_txt_file(const struct dirent *de)* проверяет, что элемент является файлом (*de->d_type == DT_REG*), оканчивающимся на *“.txt”* (вообще просто содержащим это сочетание символов в названии, но иметь его не в конце файл не может). Аналогично возвращает *0*, если условия соблюдены, и не *0* в любом другом случае.

Функция *void assign_dir(const char *path, const struct dirent *de, long long int *res)* в зависимости от названия директории вызывает функцию *add()* или *mul()* и присваивает её значение содержимому указателя res.

Функция *void add_file(const char *dir_name, const struct dirent *de, long long int *sum)* открывает файл и с помощью функции *fscanf()* считывает числа в нём, пока они не кончатся. Эти числа суммируются к содержимому указателя *sum*. Функция *void mul_file(const char *dir_name, const struct dirent *de, long long int *prod)* аналогичным образом находит произведение всех чисел, записанных в файле.

Функция *void add_dir(const char *path, const struct dirent *de, long long int *sum)* вычисляет сумму или произведение содержимого директории с именем *dir_name* с помощью функций *add()* и *mul()* и добавляет это в содержимому *sum*. Аналогично произведение всех элементов считает функция *mul_dir()*.

Функция *void process_dir(const char *dir_name, long long int *res)* открывает директорию, проверяет, что она является дочерней, и вызывает функцию *assign_dir()*.

Функции *void dir_error_check(DIR *dir) и void file_error_check(FILE *file)* служат для проверки успешного открытия соответственно директории и файла.

Функция *void write_ans(const long long int res)* записывает число *res* в файл *result.txt*.

В функции *main()* инициализируется 64-битное целое число *res* со значением *0*. Вызываются функции *process_dir()* и *write_ans()*.

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. | add<br>file.txt – 1<br>file1.txt – 2 | result.txt – 3 | Проверка программы на работоспособность. |
| 2. | add<br>file.txt –<br>file1.txt – 1 | result.txt – 1 | Проверка корректной работы программы, если какие-либо файлы пустые. |
| 3. | add<br>mul<br>file.txt – 3 2<br>add | result.txt – 6 | Проверка корректной работы программы, если какая-либо директория пуста. |
| 4. | mul<br>mul<br>mul<br>mul<br>file.txt – 1 2 3 4 | result.txt – 24 | Проверка корректной работы программы с вложенными директориями. |

**Выводы**

В ходе выполнения были изучены принципы рекурсии и рекурсивных функций, обхода файловой системы типа POSIX на языке программирования Си. Были приобретены навыки использования рекурсивных методов для обхода файловых систем, анализа содержимого текстовых файлов и директорий.

Была разработана программа, обрабатывающая рекурсивными методами информацию из различных файлов и директорий.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *pathcat(const char *path1, const char *path2);
long long int add(const char *);
long long int mul(const char *);
int is_child_dir(const struct dirent *);
int is_txt_file(const struct dirent *);
void assign_dir(const char *, const struct dirent *, long long int
*);
void add_file(const char *, const struct dirent *, long long int
*);
void mul_file(const char *, const struct dirent *, long long int
*);
void add_dir(const char *, const struct dirent *, long long int *);
void mul_dir(const char *, const struct dirent *, long long int *);
void process_dir(const char *, long long int *);
void dir_error_check(DIR *);
void file_error_check(FILE *);
void write_ans(const long long int);


long long int main() {
	long long int res = 0;
	process_dir("./tmp", &res);
	write_ans(res);
	return 0;
}

char *pathcat(const char *path1, const char *path2) {
	long long int res_path_len = strlen(path1) + strlen(path2) +
2;
	char *res_path = (char *)malloc(res_path_len * sizeof(char));
	sprintf(res_path, "%s/%s", path1, path2);

	return res_path;
}

long long int add(const char *dir_name) {
	long long int sum = 0;

	DIR *dir = opendir(dir_name);

	dir_error_check(dir);
```

```
        struct dirent *de = readdir(dir);
        while ((de) != NULL) {
            if (is_child_dir(de)) {
                add_dir(dir_name, de, &sum);
            } else if (is_txt_file(de)) {
                add_file(dir_name, de, &sum);
            }
            de = readdir(dir);
        }

        closedir(dir);

        return sum;
    }

    long long int mul(const char *dir_name) {
        long long int prod = 1;

        DIR *dir = opendir(dir_name);

        dir_error_check(dir);

        struct dirent *de = readdir(dir);
        while ((de) != NULL) {
            if (is_child_dir(de)) {
                mul_dir(dir_name, de, &prod);
            } else if (is_txt_file(de)) {
                mul_file(dir_name, de, &prod);
            }

            de = readdir(dir);
        }

        closedir(dir);

        return prod;
    }

    int is_child_dir(const struct dirent *de) {
        if (de->d_type == DT_DIR)
            return (strcmp(de->d_name, ".") != 0 && strcmp(de-
>d_name, ".."));
        else
            return 0;
    }

    int is_txt_file(const struct dirent *de) {
        if (de->d_type == DT_REG)
            return (strstr(de->d_name, ".txt") != NULL);
        else
            return 0;
    }

    void assign_dir(const char *path, const struct dirent *de, long
long int *res) {
        char *dir_name = pathcat(path, de->d_name);

        if (strcmp(de->d_name, "add") == 0) {
```

```
            *res = add(dir_name);
        } else if (strcmp(de->d_name, "mul") == 0) {
            *res = mul(dir_name);
        }

        free(dir_name);
    }

    void add_file(const char *dir_name, const struct dirent *de, long
long int *sum) {
        long long int contents;

        char *file_name = pathcat(dir_name, de->d_name);
        FILE *file = fopen(file_name, "r");

        file_error_check(file);

        while (fscanf(file, "%lli", &contents) == 1) {
            *sum += contents;
        }
        fclose(file);
    }

    void mul_file(const char *dir_name, const struct dirent *de, long
long int *prod) {
        long long int contents;

        char *file_name = pathcat(dir_name, de->d_name);
        FILE *file = fopen(file_name, "r");

        file_error_check(file);

        while (fscanf(file, "%lli", &contents) == 1) {
            *prod *= contents;
        }

        fclose(file);
    }

    void add_dir(const char *path, const struct dirent *de, long long
int *sum) {
        char *dir_name = pathcat(path, de->d_name);

        if (strcmp(de->d_name, "add") == 0) {
            *sum += add(dir_name);
        } else if (strcmp(de->d_name, "mul") == 0) {
            *sum += mul(dir_name);
        }

        free(dir_name);
    }

    void mul_dir(const char *path, const struct dirent *de, long long
int *prod) {
        char *dir_name = pathcat(path, de->d_name);

        if (strcmp(de->d_name, "add") == 0) {
            *prod *= add(dir_name);
```

```
        } else if (strcmp(de->d_name, "mul") == 0) {
            *prod *= mul(dir_name);
        }

        free(dir_name);
}

void process_dir(const char *dir_name, long long int *res) {
        DIR *dir = opendir(dir_name);

        dir_error_check(dir);

        struct dirent *de = readdir(dir);
        while ((de) != NULL) {
            if (is_child_dir(de)) {
                assign_dir(dir_name, de, res);
            }

            de = readdir(dir);
        }

        closedir(dir);
}

void dir_error_check(DIR *dir) {
        if (dir == NULL) {
            puts("Error accessing a directory...\n");
            exit(1);
        }
}

void file_error_check(FILE *file) {
        if (file == NULL) {
            puts("Error opening a file...\n");
            exit(1);
        }
}

void write_ans(const long long int res) {
        FILE *file = fopen("result.txt", "w");
        file_error_check(file);

        fprintf(file, "%lli", res);

        fclose(file);
}
```