**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 |  | Мальцев К.Л. |
| 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

1. В начале программы объявлены все необходимые функции и структуры данных.

2. В функции main() вызывается функция getResult(), которая получает результат обхода директорий "tmp/add" и "tmp/mul". Этот результат сохраняется в переменной result.

3. Результат выводится в файл result.txt с помощью функции outputDataToFile().

4. В функции getResult() поочередно проверяется наличие директорий "tmp/add" и "tmp/mul". Если они существуют, вызывается функция traversal(), которая рекурсивно обходит поддиректории и файлы внутри них.

5. В функции traversal() проверяется тип элемента (файл или директория) и соответствующим образом обрабатывается. Для директорий "add" и "mul" выполняются сложение и умножение результатов соответственно. Для файлов выполняются сложение или умножение чисел из файла в зависимости от операции.

6. Файлы с числовыми данными считываются с помощью функции getDataFromFile(), данные из них обрабатываются с помощью функции processData(), которая разделяет числа и сохраняет их в вектор.

7. Вектор чисел передается в функции sum() или mul() для выполнения соответствующей операции.

8. Рекурсивный обход директорий и файлов выполняется до тех пор, пока вся структура "tmp" не будет обработана.

9. Результат обработки итоговых данных возвращается из функции getResult() в функцию main() и выводится в файл result.txt.

Структуры данных представлены в коде для работы с вектором (Vector) и строкой (String).

1. Структура Vector:

- ll\* array - указатель на массив элементов типа ll (вероятно, это определено в другом месте в коде);

- int cap - текущая емкость массива (capacity), т.е. сколько элементов может содержать массив без перераспределения памяти;

- int size - текущий размер массива, т.е. сколько элементов фактически содержит массив.

Функции для работы с вектором Vector:

- Vector\* initVector() - функция инициализации вектора, выделяет память под структуру Vector.

- ll atV(Vector\* v, int idx) - функция получения значения элемента по индексу idx в векторе v.

- void pushBackV(Vector\* v, ll el) - функция добавления элемента el в конец вектора v. При необходимости увеличивает емкость и перераспределяет память.

- void printVector(Vector\* v) - функция печати содержимого вектора.

2. Структура String:

- char\* array - указатель на массив символов;

- int cap - текущая емкость массива символов;

- int size - текущий размер строки (количество символов).

Функции для работы со строкой String:

- String\* initString() - функция инициализации строки, выделяет память под структуру String.

- char atS(String\* s, int idx) - функция получения символа по индексу idx в строке s.

- void pushBackS(String\* s, char el) - функция добавления символа el в конец строки s. При необходимости увеличивает емкость и перераспределяет память.

- void printString(String\* s) - функция печати содержимого строки.

Обе структуры позволяют динамически увеличивать емкость и добавлять элементы (элементы вектора и символы в строку) при необходимости. Они предоставляют удобные методы для доступа к элементам и для печати содержимого.

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 |
|  | file.txt: 1  file1.txt: 1  file2.txt: 2 2  file3.txt: 7  file4.txt: 1 2 3  file5.txt: 3 -1 |  | Тест с e.moevm |

## 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 <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <regex.h>

#define NULL\_CH '\0'

#define ENDL "\n"

#define NULL\_CH\_BUFFER\_SIZE 1

#define MATCH\_GROUPS\_SIZE 1

const char\* kOutputFileName = "result.txt";

const char\* kPatternAdd = "^[a-zA-Z0-9/]\*add$";

const char\* kPatternMul = "^[a-zA-Z0-9/]\*mul$";

void throwIndexError();

typedef struct VectorNum

{

/\*\* Vector implementation

\* A standard container which offers access to

\* individual elements in any order.

\*/

long long\* array;

int capacity;

int size;

} VectorNum;

/\* Vector methods \*/

VectorNum\* vectorNumInit();

void vectorNumDelete(VectorNum\* vector\_num);

long long vectorNumAt(VectorNum\* vector\_num, int index);

void vectorNumPushBack(VectorNum\* vector\_num, long long new\_element);

void printVector(VectorNum\* vector\_num);

typedef struct String

{

/\*\* String implementation

\* Managing sequences of characters.

\*/

char\* array;

int capacity;

int size;

} String;

/\* String methods \*/

String\* stringInit();

void stringDelete(String\* string);

char stringAt(String\* string, int index);

void stringPushBack(String\* string, char new\_element);

void stringPrint(String\* string);

long long getResult();

void outputDataToFile(const char\* file\_path, long long result);

long long fileTreeTraversal(char\* dir\_path, int type, char operation,

regex\_t\* regex\_compiled\_add\_ptr, regex\_t\* regex\_compiled\_mul\_ptr);

int dirPathValidation(char\* dir\_path, regex\_t\* regex\_compiled);

int isConsidered(char\* dir\_name);

char\* getSubPath(char\* dir\_path, char\* sub\_file\_name);

String\* getDataFromFile(char\* file\_path);

VectorNum\* processData(String\* string);

long long vectorNumSum(VectorNum\* vector\_num);

long long vectorNumProduct(VectorNum\* vector\_num);

int main()

{

long long result = getResult();

outputDataToFile(kOutputFileName, result);

return 0;

}

void throwIndexError()

{

printf("Index out of range%s", ENDL);

}

long long getResult()

{

regex\_t regex\_compiled\_add;

regcomp(&regex\_compiled\_add, kPatternAdd, REG\_EXTENDED);

regex\_t regex\_compiled\_mul;

regcomp(&regex\_compiled\_mul, kPatternMul, REG\_EXTENDED);

DIR\* dir;

if ( dir = opendir("tmp/add") ) {

return fileTreeTraversal("tmp/add", DT\_DIR, '+', &regex\_compiled\_add, &regex\_compiled\_mul);

}

if ( dir = opendir("tmp/mul") ) {

return fileTreeTraversal("tmp/mul", DT\_DIR, '-', &regex\_compiled\_add, &regex\_compiled\_mul);

}

return 0;

}

void outputDataToFile(const char\* file\_path, long long result)

{

FILE\* fout = fopen(file\_path, "w");

fprintf(fout, "%lld", result);

fclose(fout);

}

long long fileTreeTraversal(char\* dir\_path, int type, char operation,

regex\_t\* regex\_compiled\_add\_ptr, regex\_t\* regex\_compiled\_mul\_ptr)

{

if (type == DT\_DIR) {

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

if (dirPathValidation(dir\_path, regex\_compiled\_add\_ptr)) {

long long sum = 0;

struct dirent\* de = readdir(dir);

while (de) {

char\* sub\_path = getSubPath(dir\_path, de->d\_name);

if (isConsidered(de->d\_name)) {

sum += fileTreeTraversal(sub\_path, de->d\_type, '+', regex\_compiled\_add\_ptr, regex\_compiled\_mul\_ptr);

}

de = readdir(dir);

}

closedir(dir);

return sum;

}

if (dirPathValidation(dir\_path, regex\_compiled\_mul\_ptr)) {

long long product = 1;

struct dirent\* de = readdir(dir);

while (de) {

char\* sub\_path = getSubPath(dir\_path, de->d\_name);

if (isConsidered(de->d\_name)) {

product \*= fileTreeTraversal(sub\_path, de->d\_type, '\*', regex\_compiled\_add\_ptr, regex\_compiled\_mul\_ptr);

}

de = readdir(dir);

}

closedir(dir);

return product;

}

}

if (type == DT\_REG) {

VectorNum\* numbers = processData(getDataFromFile(dir\_path));

if (operation == '+') {

return vectorNumSum(numbers);

}

if (operation == '\*') {

return vectorNumProduct(numbers);

}

}

return 0;

}

int dirPathValidation(char\* dir\_path, regex\_t\* regex\_compiled\_ptr)

{

regmatch\_t match\_groups[MATCH\_GROUPS\_SIZE];

return (regexec(regex\_compiled\_ptr, dir\_path, MATCH\_GROUPS\_SIZE, match\_groups, 0) == 0);

}

int isConsidered(char\* dir\_name)

{

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

}

char\* getSubPath(char\* dir\_path, char\* sub\_file\_name)

{

char\* sub\_path = (char\*) calloc(strlen(dir\_path) + strlen(sub\_file\_name) + 2 \* NULL\_CH\_BUFFER\_SIZE, sizeof(char));

strcpy(sub\_path, dir\_path);

sub\_path[strlen(dir\_path)] = '/';

strcat(sub\_path, sub\_file\_name);

return sub\_path;

}

String\* getDataFromFile(char\* file\_path)

{

FILE\* fin = fopen(file\_path, "r");

String\* line = stringInit();

signed char ch = 0;

while ((ch = fgetc(fin)) != EOF) {

stringPushBack(line, ch);

}

if (strchr(line->array, '\n')) {

\*strchr(line->array, '\n') = NULL\_CH;

}

fclose(fin);

return line;

}

VectorNum\* processData(String\* string)

{

VectorNum\* processed\_data = vectorNumInit();

char\* sep = " ";

char\* token;

token = strtok(string->array, sep);

while (token) {

vectorNumPushBack(processed\_data, atoll(token));

token = strtok(NULL, sep);

}

stringDelete(string);

return processed\_data;

}

long long vectorNumSum(VectorNum\* vector\_num)

{

long long sum = 0;

for (int i = 0; i < vector\_num->size; i++) {

sum += vectorNumAt(vector\_num, i);

}

vectorNumDelete(vector\_num);

return sum;

}

long long vectorNumProduct(VectorNum\* vector\_num)

{

if (vector\_num->size == 0) {

return 0;

}

long long product = 1;

for (int i = 0; i < vector\_num->size; i++) {

product \*= vectorNumAt(vector\_num, i);

}

vectorNumDelete(vector\_num);

return product;

}

VectorNum\* vectorNumInit()

{

return (VectorNum\*) calloc(1, sizeof(VectorNum));

}

void vectorNumDelete(VectorNum\* vector\_num)

{

free(vector\_num->array);

free(vector\_num);

}

long long vectorNumAt(VectorNum\* vector\_num, int index)

{

if (index >= 0 && index < vector\_num->size) {

return vector\_num->array[index];

}

throwIndexError();

return 0;

}

void vectorNumPushBack(VectorNum\* vector\_num, long long new\_element)

{

if (vector\_num->size + 1 > vector\_num->capacity) {

if (vector\_num->size == 0) {

vector\_num->capacity = 2;

} else {

vector\_num->capacity = vector\_num->capacity \* vector\_num->capacity;

}

vector\_num->array = realloc(vector\_num->array, vector\_num->capacity \* sizeof(long long));

}

vector\_num->array[vector\_num->size++] = new\_element;

}

void printVector(VectorNum\* vector\_num)

{

for (int i = 0; i < vector\_num->size; i++) {

printf("%lld ", vectorNumAt(vector\_num, i));

}

printf(ENDL);

}

String\* stringInit()

{

return (String\*) calloc(1, sizeof(String));

}

void stringDelete(String\* string)

{

free(string->array);

free(string);

}

char stringAt(String\* string, int index)

{

if (index >= 0 && index < string->size) {

return string->array[index];

}

throwIndexError();

return -1;

}

void stringPushBack(String\* string, char new\_element)

{

if (string->size + 1 + NULL\_CH\_BUFFER\_SIZE > string->capacity) {

if (string->size == 0) {

string->capacity = 2;

} else {

string->capacity = string->capacity \* string->capacity;

}

string->array = realloc(string->array, string->capacity \* sizeof(char));

}

string->array[string->size++] = new\_element;

string->array[string->size] = '\0';

}

void stringPrint(String\* string)

{

printf("%s", string->array);

}