**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: Рекурсия, циклы, рекурсивный обход  
файловой системы

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Бойцов В.А. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Целью работы является освоение работы с рекурсивными функциями и файловой системой, а также её 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 файлом в папке, в том числе во вложенных папках.

## 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 *<filename>.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*.

## Выполнение работы

В программе используются следующие библиотеки: *stdio.h, stdlib.h, string.h, dirent.h, sys/types.h, regex.h, stdbool.h*.

Определяются структуры *file\_data\_node* – элемент массива, хранящий в себе информацию из файла, а именно число *long long num* и строку *char\* data*, и *file\_list* -–сам массив, дополненный количеством элементов в нём *num*.

Для указания разделителя в пути файла, обозначений текущей и родительской директорий, пути корневой директории, имени выходного файла и шаблона имени рассматриваемых файлов используются константы *dir\_separator*, *current\_dir*, *parent\_dir*, *root\_dir, output\_file\_name* и *file\_extension\_pattern*.

Функция *file\_list makeList()* создаёт переменную *file\_list list*, инициализирует его поля и возвращает *list*.

Функция *void listInit(file\_list\* list, long long data\_num, char\* data)* принимает на вход указатель *file\_list\* list* на массив данных из файлов, а также данные *long long data\_num* и *char\* data*. Функция выделяет необходимую для хранения память и заполняет поля 0-го элемента массива.

Функция *void listAppend(file\_list\* list, long long data\_num, char\* data)* аналогична предыдущей, однако она записывает данные в конец массива.

Функция *int fileExtensionCheck(const char\* file\_name)* принимает на вход имя файла *const char\* file\_name* и с помощью регулярных выражений проверяет, необходимо ли обрабатывать этот файл (в соответствии с его именем). Возвращает *1*, если нужно и *0*, если нет.

Функция *void fileAnalisys(char\* file\_path, file\_list\* array)* принимает на вход путь к файлу *char\* file\_path* и указатель на массив данных *file\_list\* array*. Функция открывает файл на чтение, считывает из файла число и строку в *num* и *buf* соответственно, затем добавляет эти данные в массив с помощью *listInit()* или *listAppend()*.

Функция *void printFileList(file\_list\* list)* принимает на вход указатель на массив данных *file\_list\* list*, открывает (или создаёт) файл, куда записывает поэлементно данные из *list*.

Функция *int dataCompare(const void\* first\_elem, const void\* second\_elem)* принимает на вход указатели на два элемента массива данных и сравнивает числа из этих элементов. Необходима для функции сортировки.

Функция *void sortList(file\_list\* list)* принимает на вход указатель на массив данных *file\_list\* list*, вызывает функцию *qsort()*, которая сортирует массив по возрастанию чисел, считанных из файлов.

Функция *void cleanList(file\_list\* list)* принимает на вход указатель на массив данных *file\_list\* list*; в цикле *for()* поэлементно очищает память, задействованную массивом.

Функция *char\* makeSubPath(const char\* old\_path, const char\* addition)* принимает на вход адрес *const char\* old\_path* и следующую часть адреса *const char\* addition*. С помощью функций строк создаётся и возвращается адрес *char\* new\_path*, соответствующий *old\_path* с приписанным в конец *addition* через “*/*”.

Функция *bool isDirToCheck(dirent\* dir\_iter)* принимает на вход указатель на структуру *dirent\* dir\_iter*. Функция проверяет, является ли текущий файл в *dir\_iter* директорией и проверяет, не текущая ли это или предлежащая директория, а вложенная в текущую (т.е. имя не “*.*” или “*..*”). если это так и директорию нужно открывать для анализа, возвращается *True*, иначе – *False*.

Функция *void dirAnalisys(const char\* dir\_path, file\_list\* array)* принимает адрес директории *const char\* dir\_path* и принимает на вход указатель на массив данных *file\_list\* array*. С помощью функции *opendir()* открывает в *DIR\* cur\_dir* директорию по переданному адресу, если это успешно – считывает содержимое с помощью *readdir()* в указатель на структуру *dirent\* dir\_iter*. Затем в цикле *while()* содержимое *dir\_iter* анализируется: если тип равен *DT\_REG* и имя файла прошло проверку в *fileExtensionChecker()*, обновляется путь (создаётся временный путь *temp\_path*), затем функция *fileAnalisys()* считывает содержимое файла. Далее с помощью *isDirToCheck()* проверяется, является ли текущий файл директорией для анализа, и, если да, обновляется путь (создаётся временный путь *temp\_path*), функция *dirAnalisys()* запускается рекурсивно.

В функции *main()* создаётся массив данных из файлов *file\_list array*, запускается функция *dirAnalisys(),* затем массив сортируется с помощью *sortList()*, затем массив печатается в файл с помощью *printFileList(),* а затем память очищается с помощью *cleanList().*

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 |
|  | 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! | Пример из задания обрабатывается корректно |
|  | root/notATxtFileAtAll: 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.c: 1 Small text | 2 Simple text  3 Wow? Text?  5 So much files! | Тест на обработку файлов с расширением не .txt |

## Выводы

В результате выполнения работы были изучены тонкости работы с рекурсивными функциями и файловой системой. Было усвоено понятие рекурсия, изучена работа с файловой системой на языке Си. Была написана программа, рекурсивно обходящая все файлы и папки в данной директории.

# Приложение А Исходный код программы

Название файла: solution.c

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<dirent.h>

#include<stdbool.h>

#include<sys/types.h>

#include<regex.h>

#define BUF\_SIZE 300

typedef struct dirent dirent;

typedef struct file\_data\_node

{

long long num;

char\* data;

}file\_data\_node;

typedef struct file\_list

{

long long num;

file\_data\_node\* array;

}file\_list;

const char\* dir\_separator = "/";

const char\* current\_dir = ".";

const char\* parent\_dir = "..";

const char\* root\_dir = "./";

const char\* output\_file\_name = "./result.txt";

const char\* file\_extension\_pattern = "^.\*\\.txt$";

file\_list makeList()

{

file\_list list;

list.num=0;

return list;

}

void listInit(file\_list\* list, long long data\_num, char\* data)

{

list->num++;

list->array = (file\_data\_node\*)malloc(sizeof(file\_data\_node));

list->array[0].num=data\_num;

list->array[0].data = (char\*)malloc(sizeof(char)\*(strlen(data)+1));

strcpy(list->array[0].data, data);

}

void listAppend(file\_list\* list, long long data\_num, char\* data)

{

list->array=(file\_data\_node\*)realloc(list->array, sizeof(file\_data\_node)\*(list->num+1));

list->array[list->num].num=data\_num;

list->array[list->num].data = (char\*)malloc(sizeof(char)\*(strlen(data)+1));

strcpy(list->array[list->num].data, data);

list->num++;

}

int fileExtensionCheck(const char\* file\_name)

{

regex\_t pattern\_compiled;

regmatch\_t groups\_array[1];

regcomp(&pattern\_compiled, file\_extension\_pattern, REG\_EXTENDED);

int res=regexec(&pattern\_compiled, file\_name, 1, groups\_array, 0);

regfree(&pattern\_compiled);

return res==0;

}

void fileAnalisys(char\* file\_path, file\_list\* array)

{

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

if (fp)

{

long long num;

char buf[BUF\_SIZE];

fscanf(fp, "%Ld ", &num);

fgets(buf, BUF\_SIZE - 1, fp);

if (array->num==0)

listInit(array, num, buf);

else

listAppend(array, num, buf);

}

else

fprintf(stderr, "Error in opening file %s\n", file\_path);

fclose(fp);

}

void printFileList(file\_list\* list)

{

FILE\* fp = fopen(output\_file\_name, "w");

for(int i=0;i<list->num;i++)

{

fprintf(fp, "%Ld %s", list->array[i].num, list->array[i].data);

if (i<list->num-1)

fprintf(fp, "\n");

}

fclose(fp);

}

int dataCompare(const void\* first\_elem, const void\* second\_elem)

{

long long first\_num = ((file\_data\_node\*)first\_elem)->num;

long long second\_num = ((file\_data\_node\*)second\_elem)->num;

if (first\_num>second\_num)

return 1;

else if (first\_num<second\_num)

return -1;

else

return 0;

}

void sortList(file\_list\* list)

{

qsort((void\*)list->array, list->num, sizeof(file\_data\_node), dataCompare);

}

void cleanList(file\_list\* list)

{

for(unsigned long long i=0;i<list->num;i++)

{

free(list->array[i].data);

}

free(list->array);

}

char\* makeSubPath(const char\* old\_path, const char\* addition)

{

char\* new\_path=(char\*)malloc(sizeof(char)\*(strlen(addition)+strlen(old\_path)+2));

strcpy(new\_path, old\_path);

strcat(new\_path, dir\_separator);

strcat(new\_path, addition);

return new\_path;

}

bool isDirToCheck(dirent\* dir\_iter)

{

return dir\_iter->d\_type == DT\_DIR && strcmp(dir\_iter->d\_name, current\_dir) && strcmp(dir\_iter->d\_name, parent\_dir);

}

void dirAnalisys(const char\* dir\_path, file\_list\* array)

{

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

if (cur\_dir)

{

dirent\* dir\_iter = readdir(cur\_dir);

while(dir\_iter)

{

if (dir\_iter->d\_type==DT\_REG && fileExtensionCheck(dir\_iter->d\_name))

{

char\* temp\_path=makeSubPath(dir\_path, dir\_iter->d\_name);

fileAnalisys(temp\_path, array);

free(temp\_path);

}

if (isDirToCheck(dir\_iter))

{

char\* temp\_path=makeSubPath(dir\_path, dir\_iter->d\_name);

dirAnalisys(temp\_path, array);

free(temp\_path);

}

dir\_iter = readdir(cur\_dir);

}

}

else

fprintf(stderr, "Error in opening %s\n", dir\_path);

closedir(cur\_dir);

}

int main()

{

file\_list array = makeList();

dirAnalisys(root\_dir, &array);

sortList(&array);

printFileList(&array);

cleanList(&array);

}