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
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!	Пример из задания обрабатывается корректно
2.	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);
}
```