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 _____ Кудин А.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Для освоения методов работы с текстовыми строками в языке программирования C, рекурсивных алгоритмов и циклических структур данных, а также для углубленного понимания процессов обхода деревьев каталогов файловой системы следует выполнить такие шаги:

1. Изучить представление и операции над строками в языке C.
2. Освоить динамическое выделение и освобождение памяти для работы со строками.
3. Применить полученные знания для чтения и обработки текстовых файлов.
4. Реализовать рекурсивную функцию для обхода дерева каталогов.
5. Использовать циклические структуры для выполнения задачи сортировки строк.
6. Изучить и применить регулярные выражения для фильтрации файлов по расширению.
7. Создать структурированное решение для записи результатов обработки в выходной файл.

Задание

Вариант 3

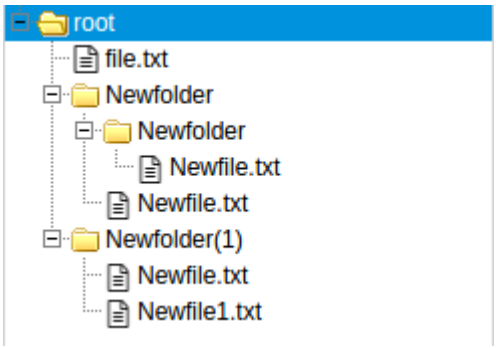
Дана некоторая корневая директория, в которой может находиться некоторое количество папок, в том числе вложенных. В этих папках хранятся некоторые текстовые файлы, имеющие имя вида <*filename*>.txt

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

Структуры данных

FileInfo: Это структура для хранения информации из файла. Она содержит два поля:

- **number**: целочисленное значение типа **long long**, предназначено для хранения числа, с которого начинается строка файла.
- **content**: указатель на **char**, который хранит саму строку, считанную из файла после числа.

FileList: Эта структура представляет собой динамический список, содержащий информацию о файлах.

- **size**: целочисленное значение типа **long long**, отражает количество элементов в списке.
- **items**: указатель на массив элементов типа **FileInfo**, хранящий информацию из всех обработанных файлов.

Функции

InitializeFileList: Эта функция инициализирует список **FileList** первым элементом. Она увеличивает размер списка, выделяет память для первого элемента и копирует в него данные.

AddToFileList: Функция добавляет новый элемент в список **FileList**. Она увеличивает размер массива **items**, выделяя дополнительную память, и добавляет в него информацию о новом файле.

CheckFileExtension: Функция проверяет, соответствует ли расширение файла заданному паттерну (в данном случае **.txt**). Используется регулярное выражение для определения соответствия.

AnalyzeFile: Функция открывает файл по указанному пути, считывает из него первое число и следующую за ним строку, затем добавляет эту информацию в список **FileList**.

OutputFileList: Функция записывает содержимое списка **FileList** в результирующий файл. Данные записываются в порядке их нахождения в списке.

CompareData: Функция сравнения, используемая функцией **qsort** для сортировки массива **items** в **FileList**. Сравнение производится на основе числа, с которого начинается строка.

SortFileList: Функция сортирует список **FileList** в порядке возрастания чисел, с которых начинаются строки.

FreeFileList: Функция освобождает память, выделенную под список **FileList**. Она освобождает память каждой строки и сам массив **items**.

CreateSubPath: Функция для создания полного пути к файлу или поддиректории на основе базового пути и дополнения (имени файла или директории).

AnalyzeDirectory: Функция рекурсивно обходит все файлы и поддиректории в заданном каталоге. Для каждого текстового файла вызывается функция **AnalyzeFile**, а для поддиректорий - рекурсивно сама себя.

Основной цикл программы (main):

В функции **main** инициализируется список **FileList**, вызывается функция **AnalyzeDirectory** для анализа директории **SEARCH_DIR**, затем список сортируется функцией **SortFileList**, после чего содержимое списка записывается в файл **RESULT_FILE** функцией **OutputFileList**, и, в конце, освобождается выделенная память с помощью **FreeFileList**

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!	OK
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!	OK

Выводы

В ходе выполнения данной работы были углублены знания и практические навыки в области программирования на языке C, особенно в части работы со строками, рекурсией и циклическими алгоритмами. Было достигнуто понимание механизмов работы с файловой системой, а также разработаны методы для рекурсивного обхода дерева каталогов с целью поиска, чтения и обработки текстовых файлов. Реализация сортировки данных, извлеченных из файлов, демонстрирует важность алгоритмов сортировки в обработке и структурировании информации.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: solution.c

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

#define BUF_SIZE 300


typedef struct {
  long long number;
  char* content;
} FileInfo;


typedef struct {
  long long size;
  FileInfo* items;
} FileList;

const char* SEARCH_DIR = "./";
const char* RESULT_FILE = "./result.txt";
const char* EXT_PATTERN = "^.*\\.txt$";


void InitializeFileList(FileList* list, long long number, char* content)
{
  list->size++;
  list->items = (FileInfo*)malloc(sizeof(FileInfo));
  list->items[0].number = number;
  list->items[0].content = strdup(content);
}


void AddToFileList(FileList* list, long long number, char* content) {
  list->items   =   (FileInfo*)realloc(list->items,   sizeof(FileInfo)   *
(list->size + 1));
  list->items[list->size].number = number;
  list->items[list->size].content = strdup(content);
  list->size++;
}


int CheckFileExtension(const char* filename) {
  regex_t regex;
  regmatch_t matches[1];
  regcomp(&regex, EXT_PATTERN, REG_EXTENDED);
  int result = regexec(&regex, filename, 1, matches, 0);
  regfree(&regex);
  return result == 0;
```

```
}


void AnalyzeFile(char* filepath, FileList* list) {
  FILE* file = fopen(filepath, "r");
  if (file) {
    long long number;
    char buffer[BUF_SIZE];
    fscanf(file, "%Ld ", &number);
    fgets(buffer, BUF_SIZE - 1, file);
    if (list->size == 0)
      InitializeFileList(list, number, buffer);
    else
      AddToFileList(list, number, buffer);
  } else {
    fprintf(stderr, "Ошибка открытия файла %s\n", filepath);
  }
  fclose(file);
}


void OutputFileList(FileList* list) {
  FILE* file = fopen(RESULT_FILE, "w");
  for(int i = 0; i < list->size; i++) {
    fprintf(file, "%Ld %s", list->items[i].number, list->items[i].content);
    if (i < list->size - 1)
      fprintf(file, "\n");
  }
  fclose(file);
}


int CompareData(const void* a, const void* b) {
  FileInfo* fa = (FileInfo*)a;
  FileInfo* fb = (FileInfo*)b;
  return (fa->number > fb->number) - (fa->number < fb->number);
}

void SortFileList(FileList* list) {
  qsort(list->items, list->size, sizeof(FileInfo), CompareData);
}


void FreeFileList(FileList* list) {
  for(unsigned long long i = 0; i < list->size; i++) {
    free(list->items[i].content);
  }
  free(list->items);
}


char* CreateSubPath(const char* base, const char* addition) {
  char* new_path = (char*)malloc(strlen(base) + strlen(addition) + 2);
  sprintf(new_path, "%s/%s", base, addition);
  return new_path;
}
```

```
void AnalyzeDirectory(const char* path, FileList* list) {
  DIR* dir = opendir(path);
  if (dir) {
    struct dirent* entry;
    while ((entry = readdir(dir)) != NULL) {
      if (entry->d_type == DT_REG && CheckFileExtension(entry->d_name)) {
        char* full_path = CreateSubPath(path, entry->d_name);
        AnalyzeFile(full_path, list);
        free(full_path);
      } else if (entry->d_type == DT_DIR && strcmp(entry->d_name, ".") &&
strcmp(entry->d_name, "..")) {
        char* full_path = CreateSubPath(path, entry->d_name);
        AnalyzeDirectory(full_path, list);
        free(full_path);
      }
    }
    closedir(dir);
  } else {
    fprintf(stderr, "Ошибка открытия директории %s\n",
path);
  }
}

int main() {
  FileList list = {0};
  AnalyzeDirectory(SEARCH_DIR, &list);
  SortFileList(&list);
  OutputFileList(&list);
  FreeFileList(&list);
}
```