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: Обход файловой системы

Студентка гр. 3383 Логинова А. Ю.

Преподаватель Гаврилов А. В.

Санкт-Петербург

2024

Цель работы

Целью данной лабораторной работы является разработка программы, выполняющая обход файловой системы и реализующая запись и чтение из файлов с расширением txt.

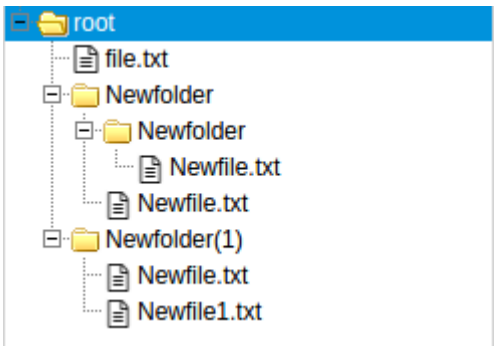
Задание

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

Для работы с текстовым файлом в программе были описаны такие структуры, как FileValue, Array_char и Array_FileValue. Две последние структуры являются массивами и имеют идентичные функции для работы с ними. Поэтому данные структуры и функции, реализующую работу с ними, были написаны с помощью макроопределения.

Структура Array_##type представляет собой динамический массив, который может изменять свой размер во время выполнения программы. Она содержит следующие поля:

type *arr;: Указатель на массив типа type, который хранит элементы массива.

size_t size;: Текущий размер массива, то есть количество элементов, которые в данный момент хранятся в массиве.

size_t capacity;: Текущая вместимость массива, то есть количество элементов, для которых выделено памяти.

void (*deleter)(type *p);: Указатель на функцию удаления, которая будет вызвана для каждого элемента при очистке массива.

Функции, связанные с этой структурой, выполняют следующие задачи:

create_##type##_array(void (*deleter)(type *)): Создает новый динамический массив с начальной вместимостью SIZE. Принимает указатель на функцию удаления в качестве аргумента. Если выделение памяти не удаётся, программа выводит сообщение об ошибке и завершает работу.

clear_##type##_array(Array_##type *arr): Очищает массив, вызывая для каждого элемента функцию удаления (если она предоставлена) и освобождает память, выделенную под массив.

resize_##type##_array(Array_##type *arr): Увеличивает вместимость массива в два раза. Если выделение памяти не удаётся, программа выводит сообщение об ошибке, очищает массив и завершает работу.

add_##type##_element(Array_##type *arr, type element): Добавляет элемент в массив. Если вместимость массива недостаточна, вызывается функция resize_##type##_array для увеличения вместимости. Затем новый элемент добавляется в конец массива, и размер массива увеличивается на 1.

append_##type##_array(Array_##type *base, type *src, size_t src_size): добавляет к массиву base массив src с помощью функции стандартной библиотеки memcpy().

Структура FileValue - пользовательский тип данных, предназначен для работы с информацией в текстовом файле:

long value_int;: поле, хранящее в себе целочисленное значение, полученное из файла.

Array_char string;: динамический массив символов, хранящий текст, полученный из файла.

Начало работы функции main начинается с создания массива символов path, который предназначен для хранения пути к директории, в которой будет производиться поиск файлов. Этот массив создается с помощью функции create_char_array, которая инициализирует динамический массив символов. Затем в массив path добавляется строка ROOT_PATH с помощью функции append_char_array, которая добавляет данные в конец массива.

Далее создается массив FileValue с помощью функции create_FileValue_array, предназначенный для хранения информации о найденных файлах. В этот массив будет добавлена информация о файлах, найденных в директории, указанной в path.

Следующим шагом является поиск файлов в указанной директории с помощью функции directory_lookup. Эта функция проходит по всем файлам в директории и добавляет информацию о каждом файле в массив FileValue.

После того как информация о файлах собрана, массив FileValue сортируется с помощью функции qsort, используя компараторную функцию compare для определения порядка элементов. Это позволяет упорядочить файлы по критерию, описанному в задании.

Сортированный массив FileValue затем записывается в файл с помощью функции write_file.

После завершения всех операций, функция освобождает память, занимаемую массивами FileValue и path, с помощью функций clear_FileValue_array и clear_char_array соответственно. Это важно для предотвращения утечек памяти.

В заключение, функция main успешно выполняет задачу поиска файлов в указанной директории, сортировки их и записи информации в файл.

Описание функций:

add_name_to_path(Array_char *path, char *name): Добавляет имя файла или директории к существующему пути. Если последний символ пути не является разделителем ('/'), добавляет его. Затем проверяет, достаточно ли места в массиве для добавления нового имени, и если нет, увеличивает вместимость массива. После этого добавляет имя к пути и завершает строку нулевым символом.

remove_name_from_path(Array_char *path): Удаляет последний элемент из пути, начиная с последнего символа разделителя ('/'). Это позволяет "вернуться" на уровень выше в директории.

compare(const void *a, const void *b): Сравнивает два элемента типа FileValue по их целочисленному значению. Используется для сортировки массива FileValue с помощью функции qsort.

get_file_extension(char* filename): Извлекает расширение файла из его имени. Возвращает указатель на начало расширения в строке имени файла или пустую строку, если расширение отсутствует.

extract_integer(char *str): Преобразует строку в целое число, используя функцию strtol. Возвращает преобразованное число.

read_next_char(Array_char *str, FILE *f): Читает следующий символ из файла и добавляет его в динамический массив символов. Если массив заполнен, увеличивает его вместимость.

read_line(Array_char *s, FILE *f): Читает строку из файла, используя функцию read_next_char, до тех пор, пока не встретит символ новой строки или конец файла.

read_txt(Array_char *path, struct dirent *entity, Array_FileValue *fileValues): Читает текстовый файл, добавляет его имя и содержимое в массив FileValue.

directory_lookup(Array_FileValue *fileValues, Array_char *path): Рекурсивно обходит директорию, ищет текстовые файлы и добавляет их в массив FileValue.

write_file(Array_FileValue *fileValues): Записывает информацию о файлах из массива FileValue в файл result.txt.

file_value_deleter(FileValue *p): Функция удаления для элементов типа FileValue, которая очищает динамический массив символов, связанный с каждым элементом FileValue.

Тестирование

Входные данные	Выходные данные
4 Where am I? 2 Simple text 5 So much files! 3 Wow? Text? 1 Small text	1 Small text 2 Simple text 3 Wow? Text? 4 Where am I? 5 So much files!

Выводы

В этой работе был изучен обход файловой системы с использованием языка программирования C. Программа реализует функционал для обхода директорий и файлов в файловой системе, начиная с заданного корневого пути. Было показано, как использовать функции для работы с директориями, такие как opendir и readdir, для перебора файлов и поддиректорий. Также было продемонстрировано чтение содержимого текстовых файлов и обработка информации из них, включая извлечение числовых значений и сохранение их в структурированном виде. В результате работы программы создается файл с результатами, отсортированными по определенному критерию.

Исходный код программы

Название файла: main.py

```
#define                                                    _DEFAULT_SOURCE
#include                                                         <stdio.h>
#include                                                        <stdlib.h>
#include                                                        <string.h>
#include                                                        <dirent.h>

#define                                SIZE                                2
#define                           ROOT_PATH                             "root"
#define                                                        Array(type)
\
                           typedef      struct      array_##type      {
\
                                                        type       *arr;
\
                                      size_t     size,     capacity;
\
                                    void     (*deleter)(type     *p);
\
                                           }            Array_##type;
\

\
      Array_##type  create_##type##_array(void  (*deleter)(type  *))  {
\
           Array_##type  arr  =  {  calloc(SIZE,  sizeof(type)),  0,  SIZE,
deleter     };                                                              \
                                                              if(arr.arr)
\
                                                           return     arr;
\
                                                             else       {
\
                printf("[create_" #type "_array()]: memory allocation
error!\n");                                                               \
                                                                  exit(1);
\
                                                                        }
\
                                                                        }
\

\
               void    clear_##type##_array(Array_##type    *arr)    {
\
                                          if      (arr->deleter)      {
\
                         for  (size_t  i  =  0;  i  <  arr->size;  ++i)
\
                                           arr->deleter(&arr->arr[i]);
\
                                                                        }
\
```

```
                                                        free(arr->arr);
\
                                                        }
\
\
                  void    resize_##type##_array(Array_##type    *arr){
\
                                              arr->capacity     *=     2;
\
        type *temp = realloc(arr->arr, arr->capacity * sizeof(type));
\
                                                          if(temp)
\
                                                arr->arr    =    temp;
\
                                                       else       {
\
                printf("[resize_" #type "_array()]: memory allocation
error!\n");                                                          \
                                                  clear_##type##_array(arr);
\
                                                          exit(1);
\
                                                          }
\
                                                          }
\
\
       void  add_##type##_element(Array_##type  *arr,  type  element)  {
\
                        if(arr->capacity   ==   (arr->size   +   1))   {
\
                                             resize_##type##_array(arr);
\
                                                          }
\
                                  arr->arr[arr->size++]    =    element;
\
                                                          }
\
\
     void append_##type##_array(Array_##type *base, type *src, size_t
src_size      )      {                                                 \
                    while(base->capacity   <   base->size   +   src_size)
\
                                           resize_##type##_array(base);
\
        memcpy(&base->arr[base->size], src, src_size * sizeof(type));
\
                                        base->size     +=     src_size;
\
                                                          }
\
```

```
Array(char)

typedef                                 struct                                  {
                                                  long          value_int;
                                             Array_char             string;
}                                                                 FileValue;

Array(FileValue)

void    add_name_to_path(Array_char    *path,    char    *name)    {
                if    (path->arr[path->size    -    1]    !=    '/')
                                        add_char_element(path,    '/');

         if((strlen(name)   +   path->size   +   1)   >=   path->capacity)
                                           resize_char_array(path);

            for(size_t   i   =   0;   i   <   strlen(name);   i++)   {
                              path->arr[path->size++]    =    name[i];
                                                                        }

                              path->arr[path->size]       =       '\0';
}

void         remove_name_from_path(Array_char         *path)         {
               char    *slash_ptr    =    strrchr(path->arr,    '/');
                                          if(slash_ptr)           {
                                         *slash_ptr     =     '\0';
                                path->size    =    strlen(path->arr);
                                                                        }
}

int     compare(const     void     *a,     const     void     *b)     {
             const    FileValue    *f_v1    =    (const    FileValue*)a;
             const    FileValue    *f_v2    =    (const    FileValue*)b;

                       if(f_v1->value_int      >      f_v2->value_int)
                                                        return       1;
                 else    if(f_v1->value_int    <     f_v2->value_int)
                                                       return       -1;
                                                                  else
                                                        return       0;
}

char*          get_file_extension(char*          filename)          {
                 char     *period     =     strrchr(filename,     '.');
                       if(period      &&      period      +      1)
                                     return     period     +     1;
                                                                  else
                                                   return       "";
}

long             extract_integer(char             *str)             {
                                           char           *endptr;
                      return      strtol(str,      &endptr,      10);
}
```

```
char      read_next_char(Array_char      *str,      FILE      *f)      {
                              char       c       =       (char)getc(f);
                    if(c     !=     EOF     &&     c     !=     '\n')     {
                                  if(str->size    +    1    ==    str->capacity)
                                                    resize_char_array(str);
                                               str->arr[str->size++]     =     c;
                                              str->arr[str->size]     =     '\0';
                                                                 return       c;
                                                                              }
                                                                            else
                                                                 return       0;
}

void       read_line(Array_char       *s,       FILE       *f)       {
                                       while(read_next_char(s,         f));
}

void read_txt(Array_char *path, struct dirent *entity, Array_FileValue
*fileValues)                                                                {
                               add_name_to_path(path,       entity->d_name);
                    FILE          *f     =     fopen(path->arr,     "r");

                                                    FileValue           f_v;
                          f_v.string      =      create_char_array(NULL);
                                         read_line(&f_v.string,          f);
                    f_v.value_int    =    extract_integer(f_v.string.arr);
                                   add_FileValue_element(fileValues,      f_v);

                                                                  fclose(f);
                                              remove_name_from_path(path);
}

void directory_lookup(Array_FileValue *fileValues, Array_char *path) {
                         DIR     *directory     =     opendir(path->arr);
                                                  if           (!directory)
          return (void)printf("[directory_lookup()]: directory in {%s}
is                          NULL\n",                           path->arr);

     for(struct dirent *entity = readdir(directory); entity; entity =
readdir(directory))                                                         {
          if(entity->d_type == DT_DIR && strcmp(entity->d_name, ".") !=
0      &&      strcmp(entity->d_name,      "..")      !=      0)      {

                                    add_name_to_path(path,   entity->d_name);
                                       directory_lookup(fileValues,   path);
                                              remove_name_from_path(path);

                         }   else   if   (entity->d_type   ==   DT_REG   &&
strcmp(get_file_extension(entity->d_name),     "txt")     ==     0)     {
                                    read_txt(path,   entity,   fileValues);
                                                                            }
                                                                            }
}

void          write_file(Array_FileValue          *fileValues)          {
```

```
                    FILE     *f     =     fopen("result.txt",     "w");

            for(int   i   =   0;   i   <   fileValues->size;   i++)   {
                                    if(fileValues->arr[i].string.arr)
                    fprintf(f,  "%s\n",  fileValues->arr[i].string.arr);
                                                                      }
                                                              fclose(f);
}

void            file_value_deleter(FileValue            *p)            {
                                          clear_char_array(&p->string);
}

int                                 main()                                {
                  Array_char     path     =     create_char_array(NULL);
           append_char_array(&path,   ROOT_PATH,   strlen(ROOT_PATH));

     Array_FileValue f_v = create_FileValue_array(file_value_deleter);

                                    directory_lookup(&f_v,         &path);
           qsort(f_v.arr,   f_v.size,   sizeof(FileValue),   compare);
                                                  write_file(&f_v);

                                         clear_FileValue_array(&f_v);
                                             clear_char_array(&path);
                                                 return             0;
}
```