**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 _____ Ягудин Д. Р.

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 названия текстовых файлов, записывает их полные пути в виде строки в файл.

**Задание**

Вариант 1

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.

Требуется найти файл, который содержит строку "Minotaur" (файл-минотавр).

Файл, с которого следует начинать поиск, всегда называется file.txt (но полный путь к нему неизвестен).

Каждый текстовый файл, кроме искомого, может содержать в себе ссылку на название другого файла (эта ссылка не содержит пути к файлу). Таких ссылок может быть несколько. Программа должна вывести правильную цепочку файлов (с путями), которая привела к поимке файла-минотавра. Цепочка, приводящая к файлу-минотавру может быть только одна. Общее количество файлов в каталоге не может быть больше 3000. Циклических зависимостей быть не может. Файлы не могут иметь одинаковые имена. 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 labyrinth

**Выполнение работы**

• Функция return_Fname принимает строку в качестве аргумента и возвращает имя файла из этой строки, если оно соответствует заданному шаблону регулярного выражения. Эта функция используется для извлечения имени файла из строки, проверяя, соответствует ли оно определённым критериям, таким как наличие допустимых символов, правильное расширение и так далее. Например, если строка содержит путь к файлу, функция проверяет, есть ли в этой строке имя файла, соответствующее шаблону, и возвращает это имя.

• Функция add_path принимает две строки, представляющие собой части пути, и объединяет их в одну строку, представляющую полный путь. Эта функция полезна для создания корректных путей к файлам и каталогам, особенно в тех случаях, когда нужно объединить базовый путь с относительным. Функция учитывает особенности операционной системы, например, правильное использование разделителей каталогов. Например, вызов add_path('C:/Users', 'Documents/file.txt') вернёт строку 'C:/Users/Documents/file.txt'.

• Функция find_F выполняет рекурсивный поиск файла с именем "file.txt" в указанном каталоге и его подкаталогах. При нахождении файла функция выполняет следующие действия:

Добавляет полный путь к найденному файлу в строку paths, что позволяет отслеживать все найденные файлы.

Читает содержимое найденного файла.

В содержимом файла ищет строку "Minotaur". Если такая строка найдена, функция записывает путь к найденному файлу в файл "result.txt".

Из содержимого найденного файла извлекает имя файла с помощью функции return_Fname и вызывает саму себя рекурсивно для обработки этого нового имени файла.

Если текущий элемент в каталоге является подкаталогом, функция рекурсивно опускается в этот подкаталог и продолжает поиск файла.

Таким образом, функция find_F обеспечивает полный обход каталога и его подкаталогов, а также обработку найденных файлов и поиск по их содержимому.

Эти функции совместно обеспечивают мощный инструмент для поиска файлов и обработки их содержимого в иерархических структурах каталогов.

**Тестирование**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | ./labyrinth | `./root/add/add/file.txt`<br>`./root/add/mul/add/file4.txt`<br><br>`./root/add/mul/file2.txt`<br><br>`./root/add/mul/file3.txt` | Тест с e.moevm |
| 2. | . | `./labyrinth/J0/n2/JU260/q1/r0/file.txt`<br>`./file3.txt`<br>`./file2.txt` | Тест, проверяющий, глубину вхождения рекурсии |

**Выводы**

В ходе выполнения данной работы были приобретены навыки эффективного использования рекурсивных методов для обхода дерева файлов, а также работы с файловой системой, анализа данных о файлах и записью информации в файл. 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: main.c

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

char* pattern = "([A-z0-9_]+\\.txt)";

char* return_Fname(char* buffer){
    regex_t regexCompiled;
    regcomp(&regexCompiled, pattern, REG_EXTENDED);
    regmatch_t groups[2];

    int j = 0;
    char* f_name = (char*)malloc(sizeof(char) * ((groups[1].rm_eo -
groups[1].rm_eo) + 1));

    if(regexec(&regexCompiled, buffer, 2, groups, 0) == 0){
        for(int i = groups[1].rm_so; i < groups[1].rm_eo; i++){
            f_name[j++] = buffer[i];
        }
        f_name[j] = '\0';
        return f_name;
    }else{
        return NULL;
    }
}

char* add_path(char* old_s, char* new_path){
    char* new_s;

    if (old_s == NULL){
        new_s = (char*)malloc(sizeof(char)*(strlen(new_path) + 1));
        strcpy(new_s, new_path);
        return new_s;
    }

          new_s   =   (char*)malloc(sizeof(char)*(strlen(old_s)   +
strlen(new_path) + 2));
    strcpy(new_s, old_s);

    int size = strlen(old_s);
    int maxsize = strlen(old_s) + strlen(new_path) + 1;
    int j = 0;

    new_s[size] = '\n';

    for(int i = size + 1; i < maxsize + 1; i++){
        new_s[i] = new_path[j++];
    }
```

```
        new_s[maxsize] = '\0';
        return new_s;
    }


    void find_F(char* f_name, char* path, char* paths){
        DIR* file_status;
        struct dirent* file;
        file_status = opendir(path);

        if(file_status == NULL){
            printf("Error, directory won't be open");
            exit(1);
        }

        while((file = readdir(file_status)) != NULL){
              if (strcmp(f_name, file->d_name) == 0 && file->d_type ==
DT_REG)
            {
                                                    char*    file_path    =
(char*)malloc(sizeof(char)*(strlen(path)+strlen(f_name) + 2));
                sprintf(file_path, "%s/%s", path, file->d_name);

                paths = add_path(paths, file_path);

                char buffer[256];
                FILE *fp = fopen(file_path, "r");

                if(fp){
                    while((fgets(buffer, 256, fp))!=NULL){
                        if(!strcmp(buffer, "Minotaur")){
                            FILE *nfp = fopen("result.txt", "w");

                            if(nfp)
                            {
                                fputs(paths, nfp);
                                fclose(nfp);
                            }
                        }else{
                            char* nameNextFile = return_Fname(buffer);

                            if (nameNextFile != NULL){
                                find_F(nameNextFile, ".", paths);
                            }
                        }
                    }
                    fclose(fp);
                }
            }
               else if(file->d_type == DT_DIR && strcmp(file->d_name,
"..") && strcmp(file->d_name, ".")){
                                                      char*    new_dir    =
(char*)malloc(sizeof(char)*(strlen(path)+strlen(file->d_name) + 2));

                sprintf(new_dir, "%s/%s", path, file->d_name);
                find_F(f_name, new_dir, paths);
            }
```

```
    }

    if(closedir(file_status) == -1){
        printf("directory don't be close");
        exit(1);
    }
}

int main(){
    char* paths = NULL;
    find_F("file.txt", "./labyrinth", paths);
    return 0;
}
```

▫▫