# PROJECT REPORT

## Project Title: File Tally

# Table of Contents

# 1.Project Overview:

## 1.1 Project Title:

**File Tally:** File and Folder Counting System

## 1.2 Purpose:

The primary objective of this project is to create a program that scans specified drives on a system, enumerates all files and folders, and provides a count of both. This project is aimed at understanding file system navigation, recursive directory traversal, and statistical reporting using C programming.

# 2. Features:

- **Drive Scanning:** The program can scan both C: and D: drives.
- **File and Folder Count:** It counts the number of files and folders recursively, including nested directories.
- **Error Handling:** It handles inaccessible drives and skips over problematic directories.
- **Cross-Platform Compatibility:** Designed primarily for Windows systems using backslashes for file paths.
- **Informative Output:** Displays a summary of total files and folders found.

# 3. Tools and Technologies Used:

- **Programming Language:** C
- **Compiler:** GCC or MinGW (for Windows)
- **Libraries:**
    - stdio.h: Standard I/O operations
    - stdlib.h: Memory allocation and process control
    - dirent.h: Directory handling
    - sys/stat.h: File status information
    - string.h: String manipulation

# 4. Code Explanation:

## 4.1 Main Components:

### 4.1.1 Function: inside(const char *path)

- **Purpose:**
  - Recursively traverses directories to count files and folders.
- **Key Operations:**
  - Opens a directory using opendir().
  - Iterates over directory entries using readdir().
  - Differentiates between files and folders using stat() and mode checking (S_ISREG, S_ISDIR).
  - Recursive call for subdirectories.

### 4.1.2 Main Function (main())

- **Purpose:**
  - Initializes the scanning process.
  - Specifies the drives to scan (C:\, D:\).
  - Calls the inside() function for each drive.
  - Handles inaccessible drives gracefully.

## 4.2 Program Flow:

1. Display a welcome message with ASCII art for better aesthetics.
2. Enumerate specified drives (C and D).
3. Count files and folders recursively using the inside() function.
4. Display the total counts of files and folders.

## 4.3 Code Snippet:

file_tally.c

C: > Users > HP > Documents > MCA > C > file_tally.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <string.h>

int dir_count = 0;
int file_count = 0;

void inside(const char *path) {
    struct dirent *entry;
    DIR *dp = opendir(path);

    if (dp == NULL) {
        // perror("opendir failed");
        return; // Unable to open directory
    }

    while ((entry = readdir(dp)) != NULL) {
        // Skip the current and parent directory entries
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0) {
            continue;
        }

        char full_path[1024];
        snprintf(full_path, sizeof(full_path), "%s\\%s", path, entry->d_name); // Use backslash for Windows

        struct stat statbuf;
        if (stat(full_path, &statbuf) == 0) {
            if (S_ISREG(statbuf.st_mode)) {
                // It's a file
                file_count++;
            } else if (S_ISDIR(statbuf.st_mode)) {
                // It's a directory
                dir_count++;
                inside(full_path); // Recursively count inside the directory
            }
        } else {
            //perror("stat failed");
        }
    }

    closedir(dp);
}

int main() {
    // Check both C: and D: drives
    const char *drives[] = {"C:\\", "D:\\"};
printf(" _______ ___ ___      _______   _______ _______ ___     ___      __   __   \n");
printf("|       |   |   |    |       | |       |   _   |   |   |   |    |  | |  |  \n");
printf("|    ___|   |   |    |    ___| |_     _|  |_|  |   |   |   |    |  |_|  |  \n");
printf("|   |___|   |   |    |   |___    |   | |       |   |   |   |    |       |  \n");
printf("|    ___|   |   |___ |    ___|   |   | |       |   |___|   |__  |_     _|  \n");
printf("|   |   |   |       |   |___    |   | |   _   |       |       | |   |    \n");
printf("|___|   |___|_______|_______|   |___| |__| |__|_______|_______| |___|    \n");


printf("\n\n File and Folder Counting System,\n Please wait till the program enumerate all the directories..... \n\n");



    for (int i = 0; i < 2; i++) {
        printf("Scanning drive %s...\n", drives[i]);
        // Check if the drive exists
        if (opendir(drives[i]) == NULL) {
            perror("Drive not accessible");
            continue; // Skip to the next drive
        }

        inside(drives[i]);
    }

    printf("The total number of files is: %d\n", file_count);
    printf("The total number of folders is: %d\n", dir_count);

    return 0;
}
```

# 5. Sample Output:


# 6. Applications:

- **File Management Tools:** Can be integrated into file management systems for summarizing folder statistics.
- **System Auditing:** Helps administrators track file and folder usage across drives.
- **Backup and Archiving:** Assists in identifying the number of files and folders for backup purposes.
- **Educational Use:** Serves as a learning tool for students studying file system operations and recursion.
- **Diagnostics:** Can be used for analyzing disk usage or troubleshooting file system-related issues.

# 7. Challenges Faced:

- **Handling Inaccessible Drives:** Used error handling (opendir() and perror()) to gracefully skip inaccessible drives.
- **Path Buffer Size:** Ensured the buffer size was sufficient for long file paths using snprintf().
- **Cross-Platform Compatibility:** Focused on Windows-specific paths with backslashes.

## 8. Learning Outcomes:

- Mastered directory traversal using the dirent.h library.
- Learned to differentiate between files and folders using stat().
- Enhanced problem-solving skills by handling recursion and edge cases.
- Gained experience in creating user-friendly terminal outputs with ASCII art and formatted messages.

## 9. Future Enhancements:

- **Cross-Platform Support:** Modify the program to support Unix-based systems by handling forward slashes (/).
- **GUI Integration:** Add a graphical interface for better user experience.
- **Custom Drive Selection:** Allow users to specify drives or directories to scan.
- **Detailed Reports:** Provide an option to save the file and folder statistics in a text or CSV file.
- **Performance Optimization:** Optimize recursion for faster directory traversal in systems with large file volumes.

## 10. Conclusion:

The File Tally project successfully demonstrates the capability of a C program to traverse a file system, count files and folders, and provide statistical results. This project not only fulfills its objectives but also serves as a foundation for more complex file system management tools.