Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report

on

"Directory Organizer with Duplicate Detection"

[Code No: COMP 202]

(For partial fulfillment of II/I Year/Semester in Computer Science/Engineering)

Submitted by

Sahaj Wagle (60)

Submitted to

Er. Sagar Aacharya

Department of Computer Science and Engineering

Abstract

This mini project presents a Directory Organizer with Duplicate Detection a command-line based system developed in C that applies core Data Structures and Algorithms concepts to automate the organization of files within a given directory. The system employs recursive directory traversal to scan files, a Hash Table with DJB2 hashing to compute and compare file content hashes for duplicate detection, and a category-based organizer that classifies files by extension and moves them into appropriate destination folders. The project is built entirely in C using a modular architecture where each DSA concept is implemented as a separate module scanner, hash table, organizer, and summary communicating through shared data structures. This project demonstrates how foundational DSA principles such as hashing, collision resolution via chaining, and recursive algorithms can solve real-world file management problems efficiently.

Keywords: Data Structures and Algorithms, Hash Table, DJB2 Hashing, Collision Resolution, Linked List Chaining, Recursive Traversal, File Organization, Duplicate Detection, Modular C Programming

Acknowledgement

I would like to express my sincere gratitude to Mr. Sagar Aacharya and the teaching staff of the Department of Computer Science and Engineering for their invaluable guidance and constructive feedback throughout the development of this mini project on Data Structures and Algorithms. I am also thankful to my institution for providing the resources and environment that enabled me to explore and apply theoretical DSA concepts in a practical C-based implementation. Finally, I am deeply grateful to my family for their constant support and encouragement.

Table of Contents

Abstract i
Acknowledgement ii
Table of Contents iii
List of Figures iv
Acronyms/Abbreviations v
Chapter 1 Introduction 1
1.1 Background 1
1.2 Objectives 2
1.3 Motivation and Significance 2
Chapter 2 Related Works 3
Chapter 3 Design and Implementation 4
3.1 System Architecture 4
3.2 DSA Components 5
3.3 System Requirement Specifications 7
Chapter 4 Discussion on the Achievements 8
Chapter 5 Conclusion and Recommendation 9
References 10
Appendix 11

List of Figures

Figure 3.1 System Architecture Diagram 4
Figure 3.2 Hash Table Structure with Chaining 6
Figure 3.3 Full Program Flow 7
Figure 5.1 CLI Landing Menu 11
Figure 5.2 Scan Output 11
Figure 5.3 Summary Output 12

Acronyms / Abbreviations

DSA	Data Structures and Algorithms
CLI	Command Line Interface
DJB2	Daniel J. Bernstein Hash Function 2
RAM	Random Access Memory

Chapter 1 Introduction

The Directory Organizer with Duplicate Detection is a command-line application developed in C that automates the task of scanning, classifying, and organizing files in a given directory. This mini project applies the fundamental principles of Data Structures and Algorithms to solve a practical and commonly encountered problem cluttered and disorganized file systems. The system scans a user-specified folder, detects duplicate files using hashing, organizes files into category-based folders by extension, and presents a summarized report of all actions taken.

1.1 Background

File systems in personal computers often accumulate large numbers of files across various categories images, documents, audio, video, and code. Manual organization of such files is tedious and error-prone. Furthermore, duplicate files consume unnecessary storage space and make retrieval difficult. Existing operating systems provide no built-in mechanism to intelligently organize or deduplicate files automatically. This project addresses that gap by applying DSA concepts specifically hash tables, linked lists, and recursive algorithms to build an efficient and lightweight file management utility entirely in C.

1.2 Objectives

- To apply recursive data structures and algorithms for directory traversal.
- To implement a hash table from scratch using DJB2 hashing and linked list chaining for collision resolution.
- To use file content hashing to accurately detect duplicate files regardless of filename.
- To design a modular C-based system where each DSA component is a separate, reusable module.
- To build a user-friendly CLI interface that ties all modules together into a single coherent workflow.

1.3 Motivation and Significance

The motivation for this project arose from the desire to apply theoretical DSA knowledge to a practical, everyday problem. The concept of a hash table one of the most important and widely used data structures finds a natural application in duplicate file detection, where a file's content can be reduced to a hash value and compared against previously seen hashes in O(1) average time. Similarly, recursive directory traversal demonstrates the practical use of recursion in navigating tree-like file system structures. By implementing these concepts from scratch in C, this project deepens the understanding of how DSA underpins real world software systems.

Chapter 2 Related Works

Several systems and tools have addressed the problem of file organization and duplicate detection, each using different approaches:

fdupes (File Duplicate Finder)

fdupes is a widely used open-source command-line utility that scans directories and identifies duplicate files by comparing file sizes and MD5 checksums. It uses a multi-pass approach: first filtering by size, then by a partial hash, and finally a full hash comparison. While effective, fdupes relies on external cryptographic libraries for MD5 computation. This project takes a simpler approach using the DJB2 algorithm implemented natively in C, making it self-contained and educational.

Python-based File Organizers

Several Python-based file organizers available on GitHub use dictionaries (Python's built-in hash map) to map file extensions to destination folders. While these tools are easy to write in Python, they do not expose the underlying DSA mechanisms. This project deliberately implements the hash table from scratch in C to demonstrate the internal workings of hashing, bucket management, and collision resolution via chaining.

Significance of This Approach

Unlike existing tools that treat file organization as a scripting task, this project treats it as a DSA problem choosing appropriate data structures (hash tables, linked lists, arrays of structs) and algorithms (DJB2 hashing, recursive DFS traversal) to build an efficient, well-structured solution in C.

Chapter 3 Design and Implementation

The system was designed with a clear modular architecture where each module corresponds to a specific DSA concept. This separation ensures that each component is independently understandable and testable.

3.1 System Architecture

The system is composed of six source files and follows a pipeline architecture where the output of one module feeds into the next:

Module	Responsibility
`file_entry.h`	Shared FileEntry struct used across all modules
`scanner.c/h`	Recursive directory traversal collects file paths, extensions, sizes
`hash_table.c/h`	DJB2 hashing + hash table with chaining duplicate detection
`organizer.c/h`	Extension-to-category mapping creates folders and moves files
`summary.c/h`	Tracks and prints a final report of all actions taken
`main.c`	CLI interface ties all modules together, handles user input

Figure 3.1: System Architecture Diagram

3.2 DSA Components

3.2.1 FileEntry Struct Shared Data Structure

The FileEntry struct is the fundamental data unit shared across all modules. It stores the full path, extension, and size of each scanned file, forming the bridge between the scanner, hash table, and organizer.

```
typedef struct {
    char path[1024];
    char extension[32];
    long size;
} FileEntry;
```

A static array of MAX_FILE (1024) FileEntry structs is allocated in main() and passed by pointer to all modules, avoiding dynamic allocation at the top level and keeping memory management simple.

3.2.2 Recursive Directory Traversal Scanner Module

The scanner uses recursive depth-first traversal to walk the directory tree. For each entry encountered, it checks whether it is a directory or a file using stat(). If it is a directory, it recurses into it. If it is a file, it extracts the extension using strrchr() and stores the FileEntry in the shared array.

```
int scan_directory(const char *path, FileEntry *files, int maxFiles)
```

The use of recursion here mirrors the tree traversal algorithms studied in DSA specifically depth-first search (DFS) on a tree structure where directories are internal nodes and files are leaf nodes. The recursion naturally handles arbitrarily deep nesting without requiring an explicit stack.

3.2.3 Hash Table Core DSA Component

The hash table is the most significant DSA component in this project. It is used to detect duplicate files by hashing the contents of each file and checking whether that hash has been seen before. The implementation involves three key design decisions:

a) DJB2 Hash Function

The DJB2 algorithm, developed by Daniel J. Bernstein, is used to hash the binary contents of each file. It reads the file byte-by-byte and computes a rolling hash:

```
hash = 5381;
while ((c = fgetc(f)) != EOF)
    hash = hash * 33 + c;
```

The initial value of 5381 and the multiplier 33 are empirically chosen constants that produce excellent distribution across a wide range of inputs, minimizing collisions. This is a non-cryptographic hash function fast and appropriate for duplicate detection purposes.

b) Hash Table Structure

The hash table consists of an array of TABLE_SIZE (1024) buckets. Each bucket is a pointer to the head of a linked list of HashNode elements. Each HashNode stores the file path, its computed hash value, and a pointer to the next node in the chain.

```
typedef struct HashNode {
    char filepath[1024];
    unsigned long hash;
    struct HashNode *next;
} HashNode;

typedef struct {
    HashNode *buckets[TABLE_SIZE];
} HashTable;
```

Figure 3.2: Hash Table Structure with Chaining

c) Collision Resolution via Chaining

When two different files produce hash values that map to the same bucket (a collision), the new node is prepended to the existing linked list at that bucket. During a lookup, the entire chain is traversed and each node's hash is compared to the incoming file's hash. This approach called separate chaining keeps insertion O(1) on average and ensures no hash value is ever silently lost due to a collision.

The slot index is computed as:

```
int slot = hash % TABLE_SIZE;
```

3.2.4 Organizer Module Extension Mapping

The organizer uses a static lookup function get_category() that maps file extensions to category names using strcmp() comparisons. Categories include Images, Videos, Audio, Docs, Code, and Others. For each non-duplicate file, the organizer creates the destination folder using mkdir() if it does not already exist (checked via stat()), then moves the file using the standard C rename() function, which performs an atomic move at the OS level.

3.2.5 Summary Module

The Summary struct accumulates counters for total scanned files, successfully organized files, detected duplicates, failed moves, and per-category counts. It is initialized before the

organize phase and updated incrementally as each file is processed. The summary_print() function presents this data in a formatted table at the end of execution.

3.3 System Requirement Specifications

3.3.1 Software Specifications

- Operating System: Windows 10/11 (with MinGW GCC)
- Compiler: GCC 14.x via MinGW-w64
- Language: C (C99 standard)
- Libraries: Standard C library (stdio.h, stdlib.h, string.h, sys/stat.h), dirent.h (Windows port by Toni Ronkko)
- Build: Manual GCC compilation gcc main.c scanner.c hash_table.c organizer.c summary.c -o main

3.3.2 Hardware Specifications

- Processor: Dual-core processor or higher
- Memory (RAM): Minimum 2 GB
- Storage: Sufficient free space for organized output folders
- Input Devices: Standard keyboard for CLI interaction

Chapter 4 Discussion on the Achievements

The Directory Organizer successfully achieved all its primary objectives. The system is able to scan arbitrarily nested directories, detect duplicate files based on content rather than name, organize files into appropriate category folders, and present a clean summary all driven by DSA concepts implemented from scratch in C.

4.1 Features

1. Content-Based Duplicate Detection

The hash table implementation correctly identifies duplicate files regardless of their filename by hashing their binary contents using DJB2. Two files with identical content but different names are correctly flagged as duplicates. Duplicate files are moved to an organized/Duplicates/ folder rather than being silently overwritten or deleted, preserving data safety.

2. Recursive Directory Traversal

The scanner recursively traverses directories of arbitrary depth, treating the file system as a tree and applying depth-first search. It correctly skips the special entries '.' and '..', handles stat() errors gracefully, and extracts file extensions for every discovered file.

3. Hash Table with O(1) Average Lookup

The hash table provides O(1) average-case insertion and lookup, making duplicate detection efficient even for large numbers of files. Collision resolution via linked list chaining ensures correctness even when multiple files hash to the same bucket. Memory for each node is dynamically allocated using malloc() and properly freed by ht_free() at program exit.

4. Extension-Based File Organization

The organizer classifies files into six categories based on extension. Destination folders are created on demand using mkdir() with an existence check via stat(). Files are moved atomically using rename(), which avoids partial copies or data loss even if the program is interrupted.

5. Modular Architecture

Each DSA component is encapsulated in its own .c and .h file pair. The FileEntry struct in file_entry.h serves as the shared data contract between modules. This design mirrors software engineering best practices and makes each module independently testable.

6. Interactive CLI

The CLI provides a menu-driven interface with three options: organize a folder, scan only, and exit. Before organizing, the user is shown how many files and duplicates were found and asked to confirm. This prevents accidental file moves and gives the user full control.

Chapter 5 Conclusion and Recommendation

The Directory Organizer with Duplicate Detection successfully demonstrates how core DSA concepts hash tables, linked list chaining, recursive traversal, and struct-based data modeling can be applied to build a practical and functional software utility in C. The project achieved all its stated objectives: recursive scanning, content-based duplicate detection via DJB2 hashing, category-based file organization, and a clean CLI interface with a summary report.

The implementation deepens understanding of how hash tables work internally particularly how the choice of hash function, table size, and collision resolution strategy affect correctness and performance. It also demonstrates how recursion naturally models tree traversal and how modular design in C leads to maintainable and readable code.

5.1 Limitations

- The DJB2 hash function, while fast and well-distributed, is not cryptographically secure. Two different files could theoretically produce the same hash (a hash collision), leading to a false duplicate detection. For a production system, a stronger hash such as SHA-256 would be preferred.
- The hash table size is fixed at 1024 buckets. For very large directories with thousands of files, the chains could grow long, degrading lookup performance toward O(n).
- File paths containing spaces cannot be entered via scanf() in the current CLI implementation.
- The system does not handle symbolic links, which could cause infinite recursion in the scanner.

5.2 Future Enhancements

- Implement SHA-256 or MD5 hashing for collision-resistant duplicate detection.
- Add dynamic hash table resizing (rehashing) when the load factor exceeds a threshold.
- Support paths with spaces by using fgets() instead of scanf() for input.
- Add a color-coded CLI output using ANSI escape codes for better readability.
- Extend the organizer to support user defined category rules via a configuration file.
- Add an undo feature that can reverse the last organization operation using a log file.

References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press. — Hash Tables, Chapter 11.

2. Bernstein, D. J. (1991). DJB2 Hash Function. Retrieved from http://www.cse.yorku.ca/~oz/hash.html

3. Ronkko, T. (2019). Dirent interface for Microsoft Visual Studio. Retrieved from https://github.com/tronkko/dirent

4. Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.

5. Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley. — Symbol Tables and Hash Tables.

Appendix

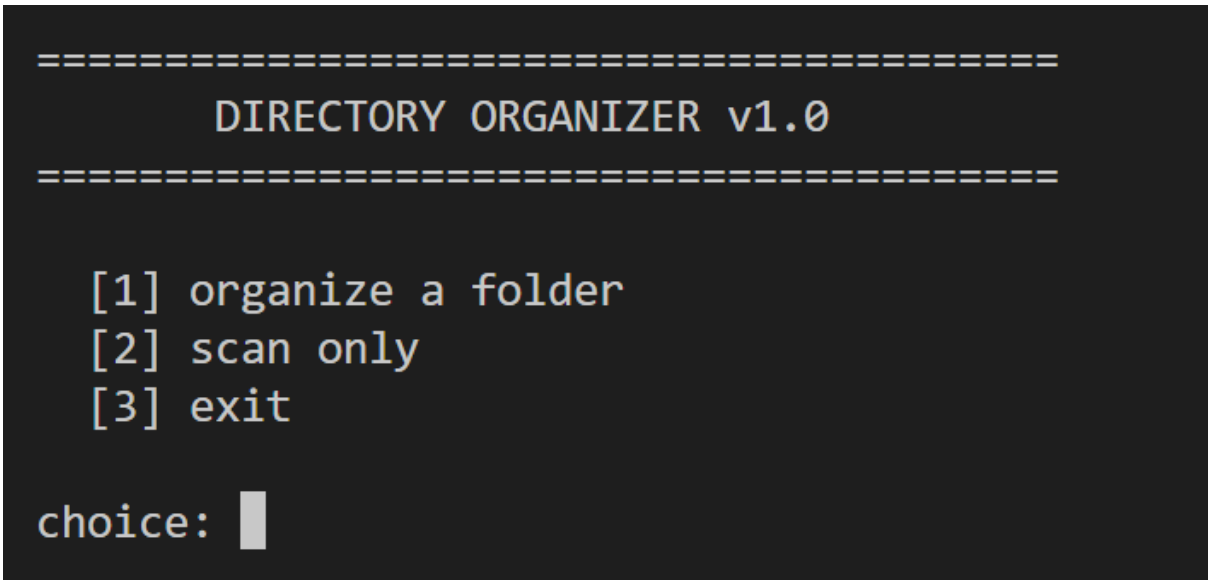

Fig 5.1: CLI Landing Menu

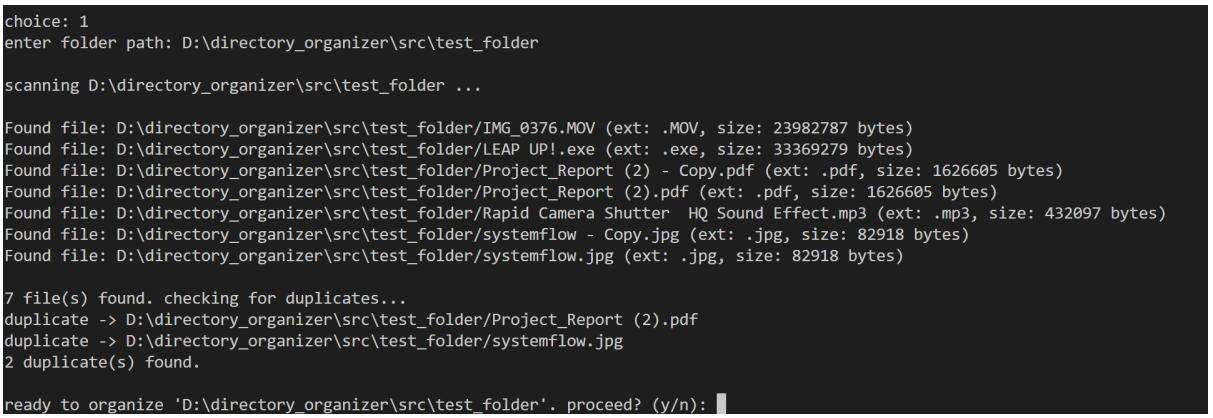

Fig 5.2: Scan and Duplicate Detection Output

```
ready to organize 'D:\directory_organizer\src\test_folder'. proceed? (y/n): y
moved IMG_0376.MOV -> organized/Others
moved LEAP UP!.exe -> organized/Others
moved Project_Report (2) - Copy.pdf -> organized/Docs
moved duplicate -> organized/Duplicates/Project_Report (2).pdf
moved Rapid Camera Shutter  HQ Sound Effect.mp3 -> organized/Audio
moved systemflow - Copy.jpg -> organized/Images
moved duplicate -> organized/Duplicates/systemflow.jpg

========================================
              SUMMARY
========================================
  total scanned  : 7
  organized      : 0
  duplicates     : 0
  failed         : 0

  categories:
    Images  : 0
    Videos  : 0
    Audio   : 0
    Docs    : 0
    Code    : 0
    Others  : 0
========================================

----------------------------------------

  [1] organize a folder
  [2] scan only
  [3] exit

choice:
```

Fig 5.3: Summary Output