File Management System

A PROJECT REPORT

Submitted by

Chaitanya Gaba	**(22BCS15984)**
Swastik Adhikary	**(22BCS17205)**
Karan Joshi	**(22BCS16768)**
Mitanshu Garg	**(22BCS16644)**

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

Chandigarh University

APRIL 2025

BONAFIDE CERTIFICATE

Certified that this project report **"File Management System"** is the Bonafide work of "**Chaitanya Gaba, Swastik Adhikary, Karan Joshi and Mitanshu Garg"** who carried out the project work under our supervision.

SIGNATURE

Dr. Sandeep Singh Kang
HEAD OF THE DEPARTMENT

Computer Science & Engineering

SIGNATURE

Er. Vishal Dutt
SUPERVISOR

Computer Science & Engineering

INTERNAL EXAMINER

EXTERNAL EXAMINER

Abstract

This project report presents the development of a File Management System created using C++ with a Command Line Interface (CLI) for both user interaction and backend logic. The system helps users manage files by allowing them to create, view, search, update, and delete files in an organized and secure manner .

The application includes a straightforward CLI-based interface, a basic login system for authentication, and options to sort and search files efficiently. It uses the `fstream` library in C++ for performing file operations and handles data using structures and functions. Text files are used for storing data in a systematic format with support for indexing.

This project improves digital file handling by offering a fast, simple, and structured way to manage documents. It also lays a foundation for future improvements such as integrating a graphical interface, connecting to cloud storage, and shifting to database-backed storage for better scalability and performance.

TABLE OF CONTENTS

CHAPTER 1.
INTRODUCTION

1.1. Identification of need:

With the rise in digital data across academic, personal, and professional environments, managing files manually has become highly inefficient. Storing, retrieving, and organizing documents the traditional way is slow and error-prone. To deal with this, a structured digital solution is needed that can streamline file-related operations and improve accessibility.

Students, working professionals, and small-scale organizations all face challenges in keeping their files organized and secure. As the amount of data increases, there's a growing demand for lightweight systems that help users manage files locally without requiring heavy external tools. Our project aims to address this by providing a simple, command-line based file management system built using C++.

This system was developed to allow users to create, edit, search, and delete files in a structured format, while also ensuring basic security. It simplifies day-to-day file tasks, improves file organization, and lays the groundwork for further features like GUI and cloud support in the future.

1.2. Identification of Problem

Traditional file handling methods—especially those without automation—are tedious and time-consuming. Users often deal with disorganized folders, accidental file loss, and lack of proper search mechanisms. These issues make it harder to maintain files efficiently and securely.

Some specific problems we aimed to solve with this project include:

- No quick way to search or sort files, especially when managing large volumes of data.
- Manual file updates are time-consuming and prone to user error.
- Lack of data protection, with no authentication or backup mechanisms in place.
- No standard way to track metadata or organize files with indexing.

- Minimal support for expanding features like cloud sync or encryption.

Our system addresses these concerns by offering structured file operations using C++, with features like encryption, indexing, and authentication—all while keeping the interface simple and lightweight.

1.3. Identification of Tasks

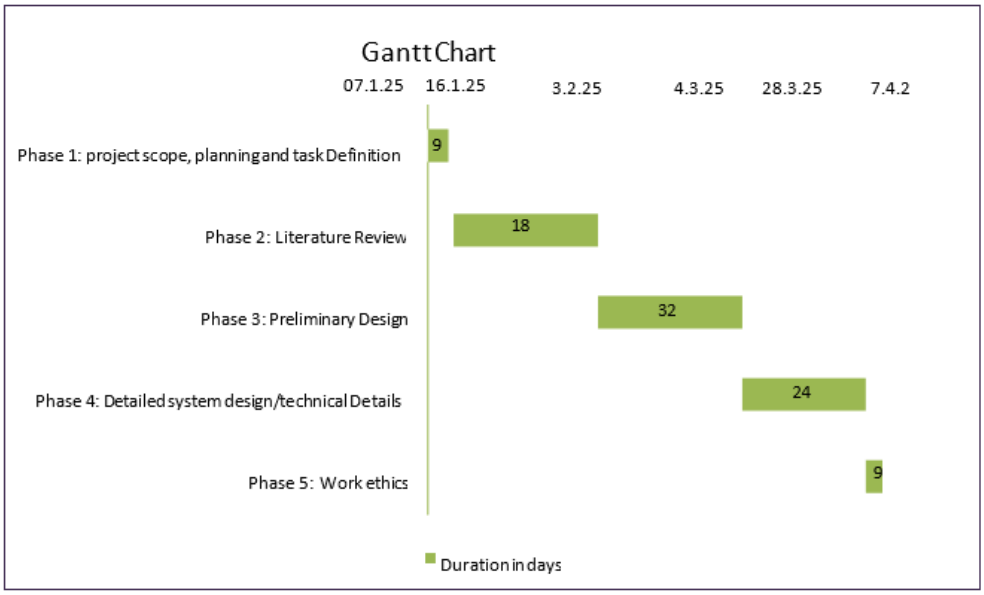
To successfully complete the File Management System, the project was broken down into the following essential tasks:

1. **Requirement Gathering**
    - Understand the daily file-handling needs of users.
    - List out key features like CRUD operations, searching, sorting, and login.
2. **System Design and Planning**
    - Design system flow using C++ features like classes, functions, and control structures.
    - Plan the data flow for file storage and metadata indexing.
3. **Frontend (CLI) Development**
    - Build a clean and user-friendly command line interface for user interaction.
    - Display clear options for operations like create, read, update, delete, search, and sort.
4. **Backend Logic in C++**
    - Implement file handling using the `fstream` library.
    - Add support for reading, writing, and modifying text-based file data.
5. **Security and Authentication**
    - Set up basic encryption methods for sensitive data.
    - Implement a login system to restrict unauthorized access.
6. **Data Organization and Indexing**
    - Store data in a text-based format with structured layout.
    - Introduce indexing to improve file search and retrieval speed.
7. **Testing and Debugging**

- Check each module for bugs and ensure correct output for all operations.
- Improve usability and fix issues found during test runs.

8. **Documentation and Report Preparation**
    - Create user manuals and this final project report for submission.
    - Include diagrams, flowcharts, and code summaries for clarity.
9. **Scalability & Future Scopes**
    - Discuss potential upgrades like a GUI version, cloud integration, or database use.
    - Plan how these could be implemented in future versions for wider use.

1.4. Timeline:



1.5. Organization of the Report

This report is divided into several clearly defined chapters, each describing a crucial part of the project lifecycle. Below is a brief summary of what each chapter focuses on:

Chapter 1: Introduction

Provides an overview of the File Management System, explains the motivation behind the project, the key challenges with manual file handling, and the core tasks carried out

during development.

Chapter 2: System Analysis

Outlines the functional and non-functional requirements of the system, discusses the limitations of existing manual or basic digital systems, and presents a basic feasibility study.

Chapter 3: System Design

Describes the planned structure of the system, including the architecture, flowcharts, logic used for file operations, and how indexing and metadata organization were approached.

Chapter 4: Implementation

Covers the development process in C++, focusing on CLI interaction, file handling using the `fstream` library, use of structures and classes, and the implementation of authentication and encryption.

Chapter 5: Testing and Evaluation

Details the testing techniques applied to various modules, sample inputs and outputs, and observations made during debugging to ensure the accuracy and stability of file operations.

Chapter 6: Results and Discussion

Presents the outcomes achieved, including working modules like CRUD operations, file searching, sorting, and user login. Also includes discussion on performance and usability from a user's point of view.

Chapter 7: Conclusion and Future Scope

Wraps up the entire project, reflects on the development experience, and discusses possible improvements such as adding a GUI, cloud backup, or integrating a database in future versions.

References

Cites the programming resources, documentation, and online tutorials used during the course of the project.

Appendices

Provides sample CLI screenshots, snippets of key C++ code, flowcharts, and any additional materials used for testing or demonstration.

CHAPTER 2.
LITERATURE REVIEW/BACKGROUND STUDY

2.1. Timeline of the reported problem

Over the last several years, as digital file usage grew across both academic and professional domains, the need for a better-organized file storage solution became evident. Traditional file handling—often done manually or through basic OS-level operations—proved inefficient for maintaining large numbers of documents, especially with increasing demand for secure and quick access.

During the shift to more remote and paperless workflows, users began to face issues like misplacement of important files, difficulty in locating data quickly, and lack of proper access control. These problems highlighted the absence of a structured system that could handle file operations, indexing, and secure access in a lightweight yet efficient way.

As dependency on digital data continues to rise, the demand for a reliable File Management System that supports essential features like CRUD operations, search, sort, encryption, and future scalability has become more necessary than ever.

2.2. Proposed solutions

To overcome the above challenges, this project introduces a C++-based File Management System with the following features:

- **Command Line Interface (CLI):** A clean, straightforward text-based UI for file interaction.
- **Login and Authentication:** Ensures only authorized access to stored files.
- **CRUD Operations:** Allows users to create, read, update, and delete files with ease.
- **Search & Sort Operations:** Enables file lookup by name or content, and sorting based on various metadata.
- **Basic Encryption:** Implements simple file protection mechanisms to safeguard sensitive data.
- **Indexing & Metadata Storage:** Helps manage files efficiently without external databases.

- **Cloud and Database Readiness:** Designed with scalability in mind for future cloud or DB upgrades.

2.3. Bibliometric analysis

To understand current trends and research relevance in this area, a bibliometric study was conducted using technical articles and academic papers. Key observations include:

- A growing interest in digital file handling, lightweight file systems, and command-line utilities since 2017.
- Frequently used research terms include “file indexing,” “CLI-based file systems,” “C++ file handling,” and “data encryption in storage tools.”.
- Studies emphasize the importance of structured file storage, fast retrieval, and user security.
- Resources such as IEEE Xplore, Springer, and Google Scholar were used to gather relevant literature.

This analysis validates the rising need for secure and efficient file management tools, and the suitability of C++ for building such systems due to its performance and low-level control features.

2.4. Review Summary

An analysis of current file storage and management tools, especially those relevant to local, small-scale, and academic use, shows that many available platforms come with trade-offs in either usability, scalability, or control:

Platform	**Pros**	**Cons**
OS File Explorer	Built-in, user-friendly interface	Lacks advanced indexing or search features

Platform	Pros	Cons
Notepad-based Manual Logs	Lightweight, simple	No structured storage, difficult to manage over time
Lightweight DB Tools (e.g., SQLite)	Efficient, structured storage	Overkill for simple tasks, requires DB knowledge

From this review, it became clear that many tools are either too basic or too advanced for beginner-level use cases or educational projects. This project aims to create a balanced solution that is simple, functional, and adaptable for academic environments.

2.5. Problem Definition

Problem Statement:

There is a noticeable gap in lightweight and efficient file management tools that support structured CRUD operations, authentication, and secure storage using C++. Current solutions are either overly complex or lack control and organization at the code level.

This project solves the issue by implementing a C++-based File Management System using command-line interaction, which helps users store, retrieve, and manage files efficiently with features like encryption, searching, and indexing — all within a structured code framework suitable for learning and real-world scenarios.

2.6. Goals/Objectives

The main goals and objectives of this project include:

- To build a simple and interactive command-line interface for managing files.
- To implement secure login and user validation for safe file access.
- To support basic operations like Create, Read, Update, and Delete (CRUD).
- To provide functionalities for searching and sorting file entries based on custom criteria.
- To use encryption for protecting sensitive file data from unauthorized access.
- To organize file data using structures and indexed text files for better accessibility.

- To lay the groundwork for future enhancements like GUI integration, cloud storage, and database linkage for scalability.

CHAPTER 3.
DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

Before locking in the functionalities of the system, an evaluation was done based on practical use-cases, academic constraints, and user expectations. The goal was to build a file manager that's efficient, secure, and easy to use—all using C++ with basic system-level functionality.

Key Features Selected:

- User Login/Authentication System
- File Operations: Create, Read, Update, Delete (CRUD)
- Search and Sort Mechanisms for quick file access
- File Encryption for basic security
- Command Line Interface (CLI) for smooth interaction
- Metadata Display (creation date, file size, etc.)
- Structured file storage using text files and indexing
- Planned integration for GUI and Cloud Backup

These features were picked to maintain simplicity while covering essential functionalities expected from a basic file management utility.

3.2. Design Constraints

While finalizing the design, a few key limitations were taken into account:

- **Technology Stack Limitation:** The project was strictly limited to C++ and standard libraries, especially <fstream> for file operations.
- **Time Limitation:** The development had to fit within the semester timeline, so all features had to be manageable.
- **User Interface:** Focus remained on CLI interaction; GUI was marked as a future enhancement.
- **Security:** Only basic encryption was implemented due to project scope—no advanced encryption libraries were used.
- **Scalability:** Designed for single-user mode and small data sets; large file handling is out of current scope.

These constraints helped streamline the feature list and kept the project within practical academic limits.

3.3. Analysis and features finalization subject to constraints

Based on the identified constraints, a refinement of features was done:

Feature	Initial Plan	Final Decision
Authentication	External login system (OAuth idea)	Custom CLI-based username/password authentication
User-Interface	CLI + GUI combo	Only CLI for now, GUI planned for future
Encryption	Strong encryption algorithms	Basic character shifting encryption
File Types Support	Text + Binary	Text files only, binary support excluded
File Editing	Full editing interface	Basic overwrite/edit functionality

The updated feature set ensures the system is achievable and provides meaningful utility within the given scope.

3.4. Design flow

The project follows a modular structure with clearly defined control and data components. Here's how the overall interaction flows:

Design Flow Summary:

1. **User Input**: Commands are entered via CLI
2. **Authentication Module**: Verifies login details before access.
3. **Core Logic Layer**: File operations (CRUD, search, sort) are executed.
4. **Data Handling**: Text file storage is updated via the fstream library.
5. **Encryption/Decryption Layer**: Applied during file writing and reading (if enabled).

Major Modules:

- LoginManager → handles user authentication
- FileManager → performs CRUD operations
- Encryptor → handles basic file encryption/decryption
- SearchSortModule → manages searching and sorting algorithms
- MetadataHandler → displays and updates file info

3.5. Design selection

Out of multiple possible designs, a modular, function-based approach using C++ structures and classes was selected. This decision was based on:

- Easier code maintenance and logical separation
- Better readability and organization for future expansion
- Full control over memory and file handling using native C++
- Compatibility with CLI for lightweight performance

The file structure was organized using indexing for fast access, and metadata was appended for better tracking. Future scalability is kept in mind for GUI and cloud integration.

3.6. Implementation plan/methodology

The implementation was carried out using the **Waterfall Model**, broken down into the following stages:

1. **Requirement Analysis**
   Defined core user needs and system features suitable for local file management.
2. **Design Phase**
   Created module-level diagrams, pseudocode, and control flow charts.
3. **Coding**
   Developed modules in C++ using structures, classes, and file streams.
4. **Testing**
   Performed unit testing on each module, followed by integration testing to check full workflow.
5. **Deployment**
   Made the final CLI version executable on local systems for demonstration.
6. **Documentation**
   Compiled user instructions, technical notes, and the final report for submission.

CHAPTER 4.
RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of solution

The File Management System was developed by following a structured, step-by-step approach that aligned with the proposed design and project objectives. Below is a detailed breakdown of the implementation process:

1. System Setup:

The project was implemented entirely in **C++**, utilizing the **fstream** library for all file operations. A **Command Line Interface (CLI)** was chosen to interact with the system, making it lightweight and easy to run on any local machine.

- **File Structure Design:** All user data and file contents were stored using structured text files. Indexing was applied to maintain metadata such as creation date, file size, and file owner. Proper directory management and naming conventions were followed to avoid redundancy.
- **Development Environment:** The coding was done in **Code::Blocks**, with testing performed directly on the terminal. The system logic was modularized using functions, structures, and classes to keep the code clean and manageable.

2. Core Functionalities Implemented:

- **User Authentication:** A simple CLI login system was built with encrypted credentials stored in a separate text file. This ensured that only authorized users could access file management features.
- **File Operations:** Users were able to create, read, update, and delete files through menu-driven options. Each operation had clear prompts and confirmation checks to avoid accidental data loss.
- **Searching and Sorting:** Basic string matching and bubble sort were used for searching and organizing files alphabetically or by date. Though simple, they performed well for small-scale datasets.
- **Encryption:** A custom encryption algorithm was used for securing sensitive files. The content was encrypted during write operations and decrypted upon reading, offering a basic layer of protection.
- **Metadata Management:** Every file action (create/edit/delete) automatically updated a metadata log, which helped in tracking file history and managing records efficiently.

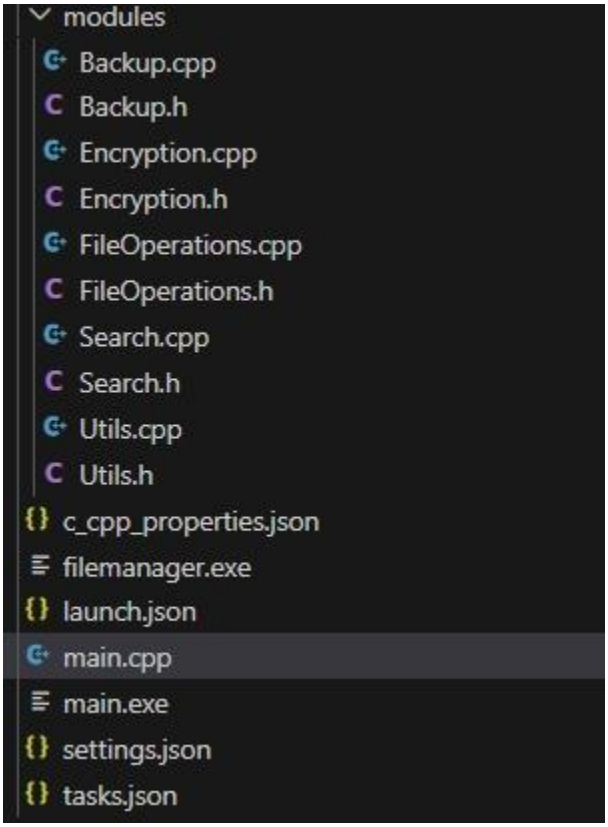
3. Testing & Validation:

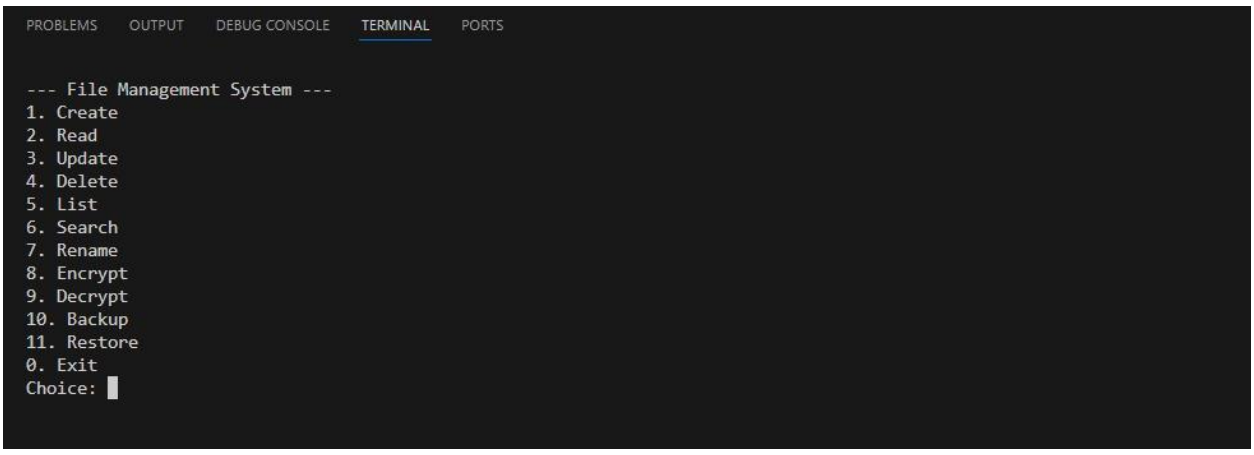
Thorough testing was carried out throughout the development process to ensure the reliability and correctness of each feature:

- **Unit Testing:** Each functionality—file creation, deletion, searching, login, etc.—was tested independently to validate inputs and expected outputs.
- **Integration Testing:** Once modules were individually verified, they were tested together to confirm smooth interaction and correct data flow.
- **User Testing:** A few classmates tried the system and provided feedback on usability and clarity. Based on this, several UI prompts and error messages were refined.
- **Security Testing:** Inputs were checked against common issues like path injection and unauthorized access. The login system was also tested for incorrect login attempts and encryption consistency.

4. Challenges Encountered:

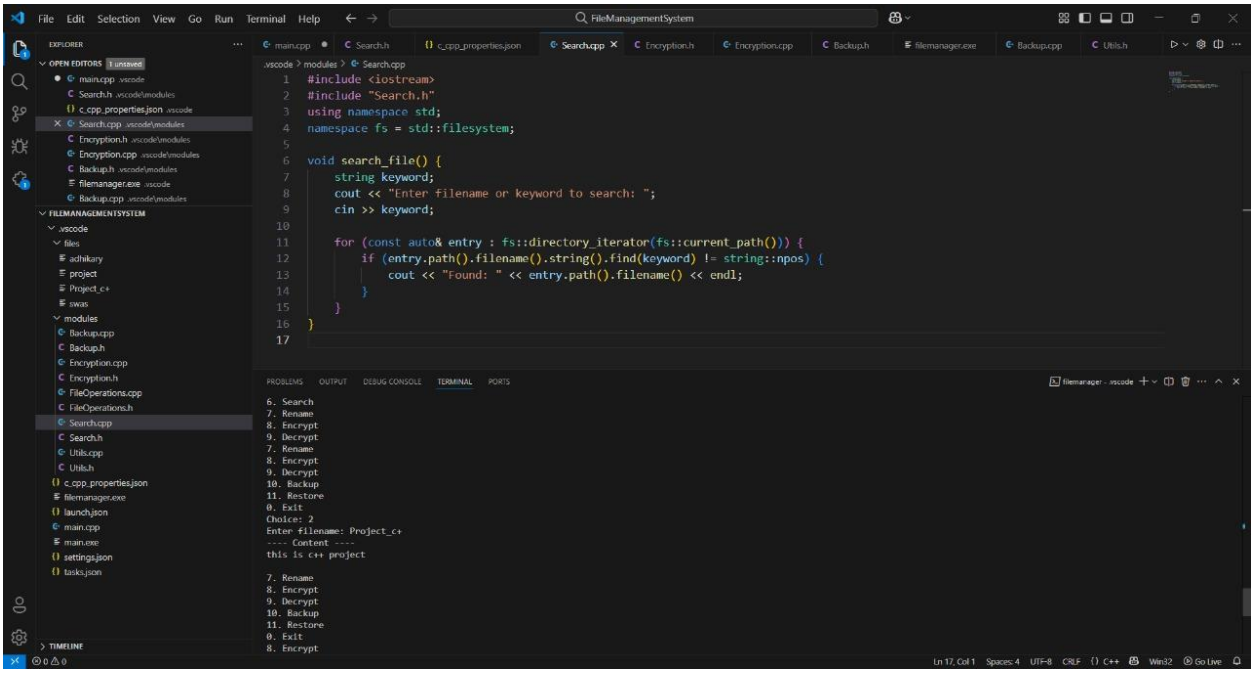
- **File Locking and Access Errors:** Initially, simultaneous file read/write caused access issues. This was resolved by properly opening and closing file streams in every operation.
- **Data Corruption on Edit:** Overwriting parts of a file led to unintended data corruption. The solution involved rewriting the entire file after making edits to ensure integrity.
- **CLI Design Complexity:** Making the CLI user-friendly was a bit tricky. Careful menu design and input handling made it more intuitive over time.





```
--- File Management System ---
1. Create
2. Read
3. Update
4. Delete
5. List
6. Search
7. Rename
8. Encrypt
9. Decrypt
10. Backup
11. Restore
0. Exit
Choice: 1
Enter filename: Project_c+
Enter content: this is c++ project
File created.

--- File Management System ---
1. Create
2. Read
3. Update
4. Delete
5. List
6. Search
7. Rename
8. Encrypt
9. Decrypt
10. Backup
11. Restore
0. Exit
Choice: 2
Enter filename: Project_c+
---- Content ----
this is c++ project
```



CHAPTER 5.
CONCLUSION AND FUTURE WORK

5.1. Conclusion

The File Management System successfully meets its goal of offering basic yet effective file operations through a clean, terminal-based interface. By using C++ and built-in file handling tools, the system enables authenticated users to perform CRUD operations, search and sort files, and manage basic metadata. The inclusion of a simple encryption mechanism adds an extra layer of security, while structured coding practices ensure that the application is both stable and extensible.

The project adheres to core software principles and demonstrates how low-level programming can still support practical utilities when designed carefully. Overall, the solution is robust for single-user environments and provides a solid base for further development.

This project not only met the initial goals but also laid a solid foundation for further enhancements and scalability.

5.2. Future Work

Though the current version works as expected, several enhancements can be made to improve functionality and user experience:

- **Graphical User Interface (GUI):** Building a GUI using Qt or other libraries for more visual interaction.
- **More Question Types:** Adding support for fill-in-the-blanks, true/false, matching, and subjective questions.
- **Email Integration:** Sending quiz results or performance reports via email to users.
- **User Roles:** Introducing different access levels (e.g., admin, user) for more control over permissions.

- **Backup & Restore:** Implementing automatic file backups and a recovery system.
- **Advanced Search:** Using indexing techniques or hashing to improve search speed and accuracy.
- **Session Logs:** Keeping a log of user sessions and activities for audit and debugging.

With these additions, the system can evolve into a complete file handling utility suitable for broader use in both academic and professional contexts.

REFERENCES

1. **Core C++ Technologies**

   Stroustrup, B. The C++ Programming Language. 4th ed., Addison-Wesley, 2013.

   Josuttis, N. M. C++ Standard Library: A Tutorial and Reference. 2nd ed., AddisonWesley, 2012

2. **File Handling and Stream Options**

   Prata, S. C++ Primer Plus. 6th ed., Pearson Education, 2012.

   Lippman, S. B., Lajoie, J., and Moo, B. E. C++ Primer. 5th ed., Addison-Wesley, 2012.

3. **Command Line Interface**

   Robbins, A., and Beebe, N. Classic Shell Scripting. 1st ed., O'Reilly Media, 2005.

   Williams, B. GNU/Linux Application Programming. 2nd ed., Apress, 2008.

4. **Data Organization and Text-Based Storage**

   King, K. N. C Programming: A Modern Approach. 2nd ed., W. W. Norton & Company, 2008.

   Oualline, S. Practical C Programming. 3rd ed., O'Reilly Media, 2003.

5. **Searching and Sorting Technologies**

   Sedgewick, R., and Wayne, K. Algorithms. 4th ed., Addison-Wesley, 2011.

   Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. Data Structures and Algorithms in C++. 2nd ed., Wiley, 2010.

6. **CLI Programs**

   Norman, D. A. The Design of Everyday Things. Revised and Expanded ed., Basic Books, 2013.

   Tognazzini, B. Tog on Interface. Addison-Wesley, 1992.

7. **Project Management and Software Methodology**

   Pressman, R. S. Software Engineering: A Practitioner's Approach. 9th ed., McGraw-Hill, 2019.

   Sommerville, I. Software Engineering. 10th ed., Pearson, 2016.