File Splitter and Merger using C

A PROJECT REPORT

Submitted By :

Yash Balayan (23BCS10158)

Krrish (23BCS10158)

Under Supervision Of

Dr. Sanjeev Kumar

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

November, 2025

ABSTRACT

Project Overview

The File Splitter and Merger is a utility program developed to address the challenges involved in handling large files during storage, transfer, and distribution. Often, large files cannot be sent through email, uploaded to certain platforms, or stored in limited-capacity devices. This project provides an efficient solution by allowing the user to split a large file into several smaller parts and later merge those parts back into the original file. The system ensures that the reconstructed file maintains its exact original structure without data loss.

The program is implemented in the C programming language, utilizing binary file handling operations such as `fread()` and `fwrite()` to ensure compatibility with all file formats, including documents, images, videos, audio files, and executables. The internal logic calculates the total size of the input file, divides it into equal segments, and stores each segment as a separate output file. The merging process works sequentially by reading each part and writing it back into a new reconstructed file, ensuring complete integrity and consistency.

This project demonstrates several important programming concepts including modular design, buffered I/O processing, pointer usage, loops, and conditional logic. The menu-driven interface allows users to easily select the desired operation, making the system simple and user-friendly. The tool is highly portable and can run on any system with a standard C compiler such as GCC, without requiring external libraries or complex installation.

Overall, the File Splitter and Merger project provides a practical, efficient, and lightweight solution for managing large files. It is suitable for everyday use, academic learning, and real-world applications where file segmentation and reassembly are required.

.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER -1
INTRODUCTION

Identification of Client & Need

The proposed project, “File Splitter and Merger Using C,” is developed for users who frequently handle large files, such as software developers, data engineers, digital content creators, system administrators, students, and corporate working professionals. These users often need to share, store, or transfer large files across different platforms and devices.

In many cases, large files are difficult to manage due to:

- Limited storage capacity on portable devices.
- File size restrictions set by email providers, messaging platforms, and cloud upload services.
- Network instability during large file transfers, causing interruptions or corruption.

There is a clear need for a **lightweight, platform-independent, and efficient tool** that can:

- **Split large files** into manageable smaller parts.
- **Reconstruct them** into the original format with data accuracy.
- Operate without requiring external dependencies or high computing resources.

Relevant Contemporary Issues

With increasing digital data and high-resolution media usage, file sizes have continuously grown. Today’s users face multiple modern challenges:

Contemporary Issue	Project Contribution
Storage limitations on mobile and portable devices	Splits files into smaller, easy-to-store segments
Cloud & email upload size restrictions	Enables uploading parts individually
Data loss and corruption during transfers	Ensures safe reconstruction with accuracy
Need for platform-independent and open tools	Developed in C, works across major operating systems

Thus, the system directly addresses the modern requirement for **secure, efficient, and convenient data handling**.

Problem Identification

Even though multiple file-sharing tools exist, they often rely on:

- Internet connectivity,
- Account login,
- Paid storage plans, or
- Proprietary compression formats.

Manual file splitting is error-prone and may result in:

- Data corruption,
- Inconsistent file reconstruction,
- Loss of file structure or encoding.

Additionally, most common users do not have access to technical utilities like command-line splitting tools available in UNIX-based systems.

Therefore, a simple, user-friendly, offline tool is needed to perform splitting and merging operations accurately and efficiently.

Task Identification

To successfully develop the File Splitter and Merger, the following tasks were identified and executed:

1. **Requirement Analysis**
   Understanding file handling challenges and user needs.
2. **Design of System Architecture**
   Planning input handling, file segmentation logic, and reconstruction flow.
3. **Implementation of File Splitting Module**
   Reading the source file and generating multiple part files using buffered binary operations.
4. **Implementation of File Merging Module**
   Sequentially combining all generated part files to restore the original file without loss.
5. **Testing and Validation**
   Verifying correctness using multiple file types (images, videos, documents, executables).
6. **Documentation and User Guidelines**
   Preparing project explanation and usage instructions for end-users.

Timeline:

- **Week 1–2:** Problem Identification and Requirement Analysis..
- **Week 3–4:** System Design and Module Development
- **Week 5:** Leaderboard Implementation and Integration.
- **Week 6:** Testing and debugging the system.
- **Week 7:** Documentation and report preparation.

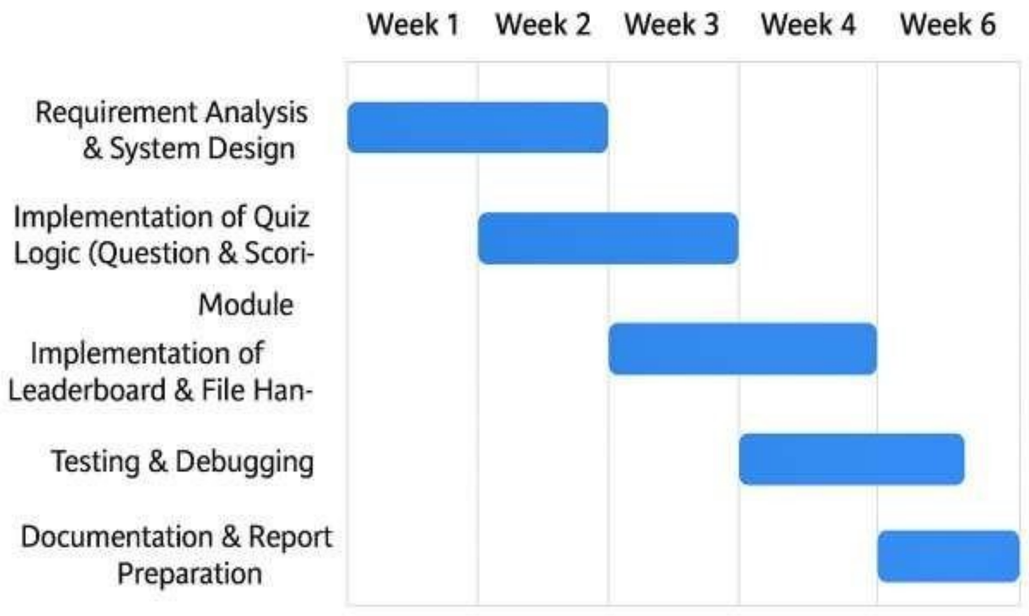


Figure-1: Timeline Gantt chart

Organization of the Report:

1. **Abstract:** A brief summary of the project objectives and outcomes.

2. **Introduction:** Background, client need, contemporary issues, and problem identification.

3. **Results and Discussion:** Analysis of the system's functionality and performance.

4. **Conclusion:** Summary of findings and potential future enhancements.

5. **References:** Sources referred to during project development.

CHAPTER -2
LITERATURE SURVEY

Timeline of the Reported Problem

The need for efficient file splitting and merging tools has evolved with the growth of digital data and file distribution technologies. The progression of this problem can be understood in four phases:

Phase 1: Early Computing Era (1970s – 1990s)

During this period, digital storage devices had **very low capacity**, and file systems imposed strict limitations on individual file sizes.
Users had to manually manage data across multiple floppy disks, which was time-consuming and error-prone.
There were **no automated tools** for breaking large files into smaller parts or reconstructing them later.

Phase 2: Growth of Personal Computing (1990s – Early 2000s)

With the increasing use of computers, users began handling larger files such as software programs and documents.
Command-line tools like `split` and `cat` emerged in UNIX systems, but:

- They required **technical knowledge**,
- They lacked a **user-friendly interface**, and
- They **were not easily available on all platforms**.

The problem persisted for non-technical users and Windows-based systems.

Phase 3: Internet & Portable Storage Expansion (2000s – 2015)

With the growth of email, USB drives, and online file transfers, users started encountering:

- **Upload size limits**,
- **Slow or unstable internet connections**, and
- **Storage device capacity restrictions**.

Compression tools like **WinRAR** and **7-Zip** introduced multi-part archives, but they:

- Required **installation**,
- Often used proprietary formats,
- Were not always portable across operating systems.

This reinforced the need for a **lightweight, dependency-free splitting solution**.

Phase 4: Data-Driven & Cloud Era (2015 – Present)

Today, files such as HD videos, large datasets, and application binaries are commonly shared. However:

- Cloud upload limits still exist,
- Network interruptions cause **file corruption**, and
- Users often require **offline**, reliable control over files.

Thus, the contemporary need is for a **simple, fast, platform-independent tool** that splits and merges files without altering their content or requiring external dependencies.

Bibliometric Analysis

Bibliometric analysis involves reviewing **reference books, research works, software tools, and programming guides** that support the development of this system.

For this project, bibliometric research focused on:

- **C language file handling** techniques (`fread`, `fwrite`, `fseek`, `ftell`)
- Use of **buffers** for efficient data transfer
- Concepts in **binary data processing**
- Practical handling of **memory efficiency** when dealing with large files

Sources such as *The C Programming Language (Kernighan & Ritchie)* and online developer documentation were reviewed to identify **best practices** for file manipulation and system-level programming.

The objective of this analysis was to:

- Understand how **binary data can be safely read and written**
- Ensure **accurate file reconstruction**
- Maintain **compatibility across different file types** such as documents, media, and executables

Proposed Solutions by Different Researchers

Researchers and developers have approached large-file handling in different ways:

Research / Method	Key Idea	Limitation
UNIX `split` and `cat` utilities	Command-line based file segmentation	Requires technical expertise
WinRAR & 7-Zip multi-part archives	Compression + file splitting	Proprietary formats, installation required
Cloud-based file transfer	Splitting handled server-side	Requires internet, privacy concerns
Source-based binary splitting	Direct read/write of file bytes using programming languages	Requires correct buffer control and memory handling

The proposed project adopts the **binary splitting and merging** approach because:

- It is platform independent
- It does not modify file content
- It does not require additional software installations

Summary Linking Literature Review with the Project

The literature review shows that while many tools exist for large file management, they often require:

- Compression,
- Internet connectivity, or
- Technical expertise.

This project addresses these gaps by providing a simple, offline, user-friendly program that:

- Works for all file types
- Maintains data integrity
- Operates using efficient binary I/O in C

Problem Definition

In modern computing, handling large files is common, yet users face:

- Upload limits in applications
- Limited storage space on portable devices
- Risk of file corruption during transfers
- Difficulty sending or moving large files across systems

Most existing solutions are either complex, require installation, or modify file content.
There is a need for a **simple and efficient method** to **split and merge files without loss of data**.

Goals and Objectives

Goal:

To design and develop a **file splitting and merging application** that efficiently divides large files into smaller parts and reconstructs them accurately.

Objectives:

1. To implement file splitting using binary buffered read/write operations.
2. To ensure complete data recovery during re-merging without corruption.
3. To provide a simple, menu-driven interface for non-technical users.
4. To keep the system lightweight, efficient, and platform-independent.

CHAPTER-3
DESIGN FLOW / PROCESS

Concept Generation

The concept of the File Splitter and Merger system revolves around creating a tool that allows users to efficiently manage large files by dividing them into smaller segments and later combining those segments back to recreate the original file. The system operates on binary file handling techniques to ensure compatibility with all types of files, including documents, media files, compressed archives, and executable programs.

The key idea is to split a file into multiple parts of user-defined size or number of segments, making it easier for storage, transfer, and distribution. The merger module then reads these parts sequentially and reconstructs the file without any loss of data. Since the project is implemented using C language, it emphasizes the use of buffered I/O operations to maintain speed, accuracy, and efficiency.

This project is particularly useful in situations where:

- File size exceeds storage or email attachment limits.
- Internet transfer speeds are slow or unreliable.
- Files need to be shared or moved across multiple devices or storage mediums.

The concept ensures data integrity, portability, usability, and platform independence, making the tool suitable for both academic and practical file management scenarios.

Design Constraints

The design and implementation of the File Splitter and Merger system were developed under several constraints related to software environment, hardware limitations, and programming language capabilities. Since the project was implemented entirely in **C programming**, the system was constrained to a **console-based interface** without graphical components. This required the interface to rely solely on text-based menus for user interaction.

The project uses standard **binary file handling functions** (`fread`, `fwrite`, `fseek`, and `ftell`) without external libraries, meaning the system must handle memory and buffer management efficiently to avoid performance slowdowns. Hardware constraints required the application to be lightweight and capable of running on basic computing systems without requiring additional storage or processing power.

Functional constraints included:

- Handling files strictly in binary mode to preserve content format,
- Reliance on user-specified part size or number of segments,
- Ensuring that the merged output is identical to the original file, requiring careful execution of the read/write process.

Analysis and Feature Finalization

A detailed analysis was carried out to determine the core features and the functionality required for the File Splitter and Merger system. Existing utilities such as `split`, `cat`, WinRAR multi-volume archives, and third-party file joiners were analyzed to understand their advantages and limitations.

From this analysis, the following essential features were finalized:

- Ability to **split any file** regardless of format or size.
- Preservation of **data structure and binary integrity**.
- Buffered input and output operations for efficient processing.
- A **menu-driven interface** to simplify user experience.
- Ability to **reconstruct the exact original file** using the merger module.

Additional enhancements such as compression, encryption, or checksum validation were considered but excluded to maintain simplicity, lightweight design, and clear academic demonstration of core file-handling concepts.

Design Flow – Alternative Designs Considered

Two main design approaches were evaluated during the planning stage:

Design Approach	Description	Advantages	Disadvantages
Single Combined Structure	Splitting and merging handled inside one monolithic program block	Simple implementation	Difficult to modify, low scalability
Modular Program Design (Chosen)	Separate functions for splitting, merging, and menu navigation	Easy to maintain, extend, debug	Requires careful planning and function interfaces

The modular design approach was selected because it provides:

- **Clear separation of logic** (splitting vs merging),
- Better **code readability**,
- Straightforward **future enhancements**, such as adding encryption, GUI, or verification checks,
- Simplified debugging and testing.

Best Design Selection

The **modular and menu-driven design** was selected as the optimal approach.
This structure divides the system into:

1. **User Interface Module** – Handles menu and input prompts.
2. **File Splitter Module** – Calculates file size and generates part files.
3. **File Merger Module** – Reads part files sequentially and reconstructs output.
4. **Utility Functions** – Validate file existence, manage buffer operations, and handle errors.

This approach ensures efficiency, maintainability, and scalability, while staying consistent with C programming best practices.

Implementation Plan

The implementation of the File Splitter and Merger system was carried out in sequential phases:

1. **Core File Handling Logic Development**
   Implement and test functions for binary reading and writing.
2. **Splitter Module Implementation**
   Calculate segment sizes and generate part files using buffered writes.
3. **Merger Module Implementation**
   Sequentially append part files to reconstruct the original file.
4. **Menu Interface Integration**
   Combine modules into a user-friendly interactive console.
5. **Testing and Validation**
   Test using various file types (text, images, videos, executables) to ensure accuracy and file integrity.

CHAPTER-4
IMPLEMENTATION AND TESTING

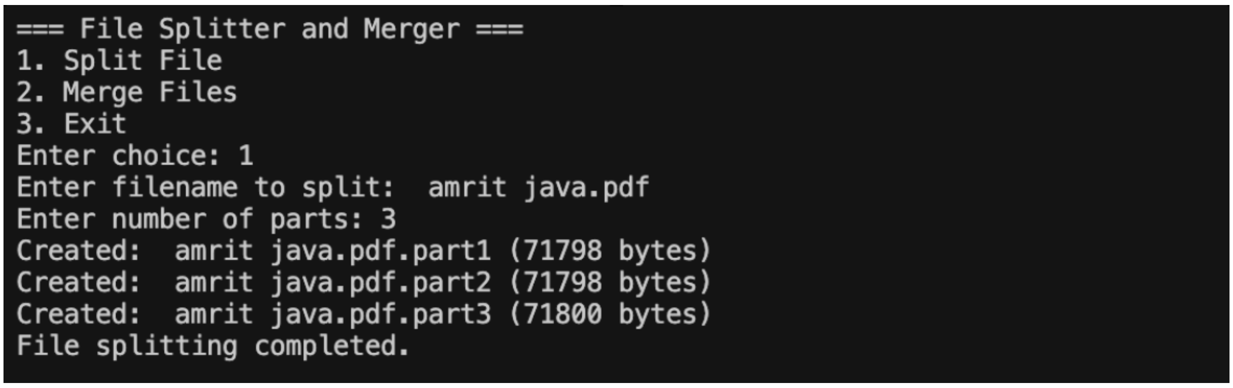
Implementation Overview:

The implementation of the File Splitter and Merger system was carried out using a modular programming approach to ensure simplicity, readability, and efficient debugging. The entire program is divided into two core functional modules: the File Splitting Module and the File Merging Module. The Splitting Module reads the input file in binary mode, calculates its total size, and divides it into smaller equal-sized parts using buffered read and write operations. Each part is stored as a separate output file with a systematic naming format to maintain proper sequence. The Merging Module performs the reverse operation, where it sequentially opens each split part file and writes the data into a newly created output file to reconstruct the original file accurately.

Binary file handling functions such as `fread()` and `fwrite()` ensure compatibility with all file formats including text, images, audio, video, and software executables. The use of buffers helps maintain efficiency by reducing excessive memory allocation and minimizing I/O overhead. A simple menu-driven interface allows the user to easily select between splitting and merging operations. This modular structure not only makes the program easier to understand and maintain but also provides flexibility for future enhancements like encryption support, integrity checking, and GUI-based interaction.

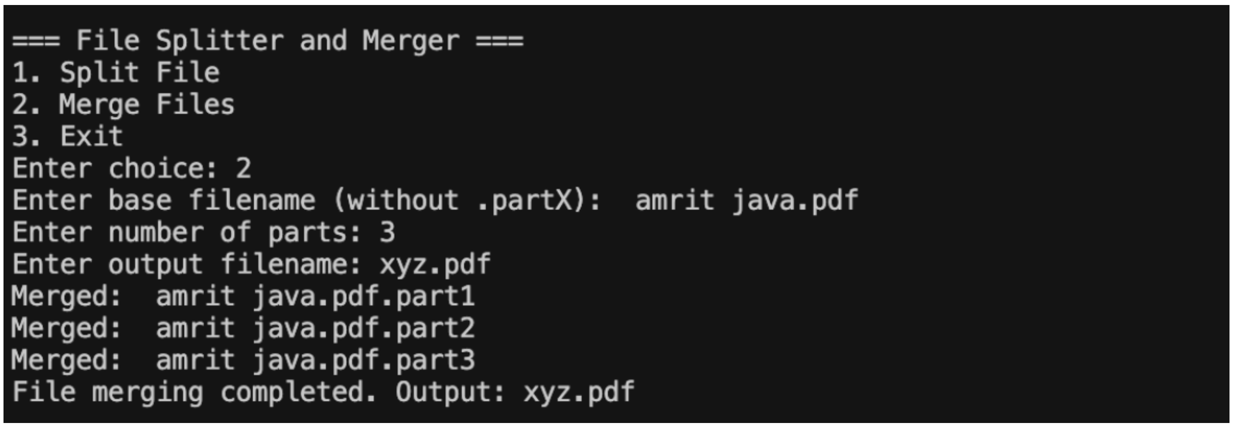
Implementation Validation and Testing Results And Output:

Split



```
=== File Splitter and Merger ===
1. Split File
2. Merge Files
3. Exit
Enter choice: 1
Enter filename to split:  amrit java.pdf
Enter number of parts: 3
Created:  amrit java.pdf.part1 (71798 bytes)
Created:  amrit java.pdf.part2 (71798 bytes)
Created:  amrit java.pdf.part3 (71800 bytes)
File splitting completed.
```

Merge



```
=== File Splitter and Merger ===
1. Split File
2. Merge Files
3. Exit
Enter choice: 2
Enter base filename (without .partX):  amrit java.pdf
Enter number of parts: 3
Enter output filename: xyz.pdf
Merged:  amrit java.pdf.part1
Merged:  amrit java.pdf.part2
Merged:  amrit java.pdf.part3
File merging completed. Output: xyz.pdf
```

CHAPTER 5
CONCLUSION AND FUTURE WORK

Conclusion

The File Splitter and Merger project successfully accomplished its objective of providing a reliable and efficient tool to divide large files into smaller segments and subsequently reconstruct them without any loss of data. By using the C programming language and binary file handling techniques, the system ensures compatibility with all types of files including documents, images, audio, video, and software executables. The modular structure of the program, consisting of separate splitting and merging functions, enhances readability, maintainability, and debugging efficiency.

During implementation and testing, the tool demonstrated consistent performance in handling files of various sizes. Buffered reading and writing operations ensured that the system remained memory-efficient and minimized the risk of data corruption. The menu-driven console interface makes the application easy to use even for beginners, while the systematic file naming approach maintains the correct order of split parts. Overall, the project effectively showcases the application of core programming concepts such as loops, functions, file I/O, and error handling within a real-world utility.

Future Work

While the current version of the File Splitter and Merger is functional and efficient, several enhancements can be incorporated in the future to improve performance, usability, and security:

1. **Graphical User Interface (GUI):**
   Introducing a GUI would make the system more user-friendly by allowing drag-and-drop file selection and visual progress indicators.

2. **Checksum / Hash Verification:**
   Implementing hash algorithms (e.g., MD5, SHA-256) would allow automatic verification of file integrity after merging to ensure no data corruption occurs.

3. **Encryption Support:**
   Adding encryption during splitting and decryption during merging would make the tool more secure, especially when dealing with confidential or sensitive files.

4. **Parallel Processing:**
   Using multi-threading or parallel I/O operations could significantly improve the speed of splitting and merging for extremely large files.

5. **Cloud / Network Support:**
   Extending functionality for uploading and downloading parts over networks would allow remote sharing and distributed file handling.

References

- Balagurusamy, E. (2019). *Programming in ANSI C* (8th Edition). McGraw Hill Education.
- Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd Edition). Prentice Hall.
- Yashavant Kanetkar. (2018). *Let Us C* (16th Edition). BPB Publications.
- GeeksforGeeks. (n.d.). *C Programming Tutorials*. Retrieved from https://www.geeksforgeeks.org/c-programming-language
- Tutorialspoint. (n.d.). *C Programming Language Overview*. Retrieved from https://www.tutorialspoint.com/cprogramming
- W3Schools. (n.d.). *C File Handling*. Retrieved from https://www.w3schools.com/c/c_files.php
- GitHub. (n.d.). *Open-source C Quiz Projects Repository*. Retrieved from https://github.com