# Mini Project – Report -Semester 3

## DATA STRUCTURES LAB - 15B17CI371

****

FILE ANALYZER AND UTILITY SYSTEMS

### TEAM MEMBERS

#### ALOKIK GARG – 22103302

#### SHIVAM ARYAN – 22103298

#### MANAV UPADHYAY – 22103278

#### SIDDHARTH AGRAWAL – 22103279

### Submitted To

#### Dr. Anuja Shukla

(Assistant Professor)

(Jaypee Institute of Information Technology)

# Problem Statement:

The existing landscape lacks a robust and user-centric file management system that caters to the multifaceted needs of users in handling files efficiently. Current file management solutions lack features such as seamless undo/redo functionalities, detailed file analysis, and an intuitive interface for user convenience. The absence of a comprehensive system poses challenges in managing and manipulating files effectively.

### Issues and Challenges:

1. **Limited Functionality:** Current file management systems lack advanced functionalities like undo/redo operations, making it challenging for users to backtrack changes effectively.
2. **Absence of Analysis Tools:** There's a gap in systems that provide insights into file contents, such as word count and detailed version histories, hindering efficient file analysis.
3. **Usability Concerns:** Many existing systems lack an intuitive interface, making file management cumbersome and time-consuming for users.

### Key Requirements:

* **File Manipulation:** Create, edit, and delete files within the system.
* **Undo/Redo Functionality:** Enable users to revert changes and reinstate previously undone actions seamlessly.
* **Analysis Capabilities:** Provide insights into file contents, such as word count and detailed version histories.
* **User-Friendly Interface:** Ensure an intuitive and easy-to-use interface for efficient file management.

## Objective:

The objective is to develop a comprehensive file management system that addresses the limitations of current solutions. This system aims to offer a suite of functionalities allowing users to manage files efficiently, perform in-depth analysis, and have the flexibility to backtrack changes effortlessly. The system will prioritize user experience by implementing an intuitive interface and robust functionalities to cater to a diverse range of file management needs.

Overall, the creation of this file management system seeks to bridge the existing gaps by offering a feature-rich, user-centric, and efficient solution for effective file manipulation and analysis.

# Requirement Specifications:

## Functional Requirements:

1. **File Operations:**
   * **Add File:** Users can add new files to the system.
   * **Remove File:** Ability to delete files from the system.
   * **Edit File:** Modify file content by replacing or appending data.
2. **Undo/Redo Functionality:**
   * **Undo Edit:** Capability to revert back to the previous state of a file.
   * **Redo Edit:** Ability to reapply undone changes to a file.
3. **Display and Analysis:**
   * **Display Files:** Users can view a list of all files present in the system.
   * **Display File Data:** Retrieve and showcase file content.
   * **Analyze File Data:** Provide word count and detailed version history for files.
   * **Compare Files:** Ability to compare content between two files.
4. **Advanced Functionalities:**
   * **Rename File:** Change the name of an existing file in the system.
   * **Auto-Complete:** Offer suggestions based on partial word entries in a file.
   * **Advanced Search:** Allow users to search for patterns using regular expressions in file content.

### Non-Functional Requirements:

1. **Efficiency:**
   * System should handle file manipulations efficiently, ensuring quick response times.
   * Minimize memory usage for storing file data and system configurations.
2. **User Interface:**
   * Intuitive and user-friendly interface for seamless interaction with the system.
   * Clear and informative output for file content, version history, and analysis results.
3. **Reliability and Error Handling:**
   * Error messages and prompts for incorrect user inputs or file access failures.
   * Robust handling of file operations to prevent data corruption or loss.
4. **Security:**
   * User authentication to access the file management functionalities.
   * Ensuring file integrity and data privacy within the system.
5. **Scalability:**
   * Design system to accommodate potential future enhancements and increased file volumes.

### System Environment:

* **Languages Used:** C++.
* **Libraries:** Standard libraries for file I/O, string manipulation, data structures.
* **IDE:** Development in any compatible C++ integrated development environment.
* **Operating System:** Compatible across multiple operating systems supporting C++.

### Constraints:

* **File Size:** Handle file sizes within the limitations of the system memory.
* **Authentication:** Ensure secure user authentication for accessing the file management system.
* **Compatibility:** Maintain compatibility with various file formats and structures.

### Assumptions:

* Users are familiar with basic file operations and system navigation.
* The system will be deployed in an environment that supports C++ execution.

Overall, the system aims to provide a comprehensive set of functionalities while meeting efficiency, usability, and security standards in managing and analyzing files.

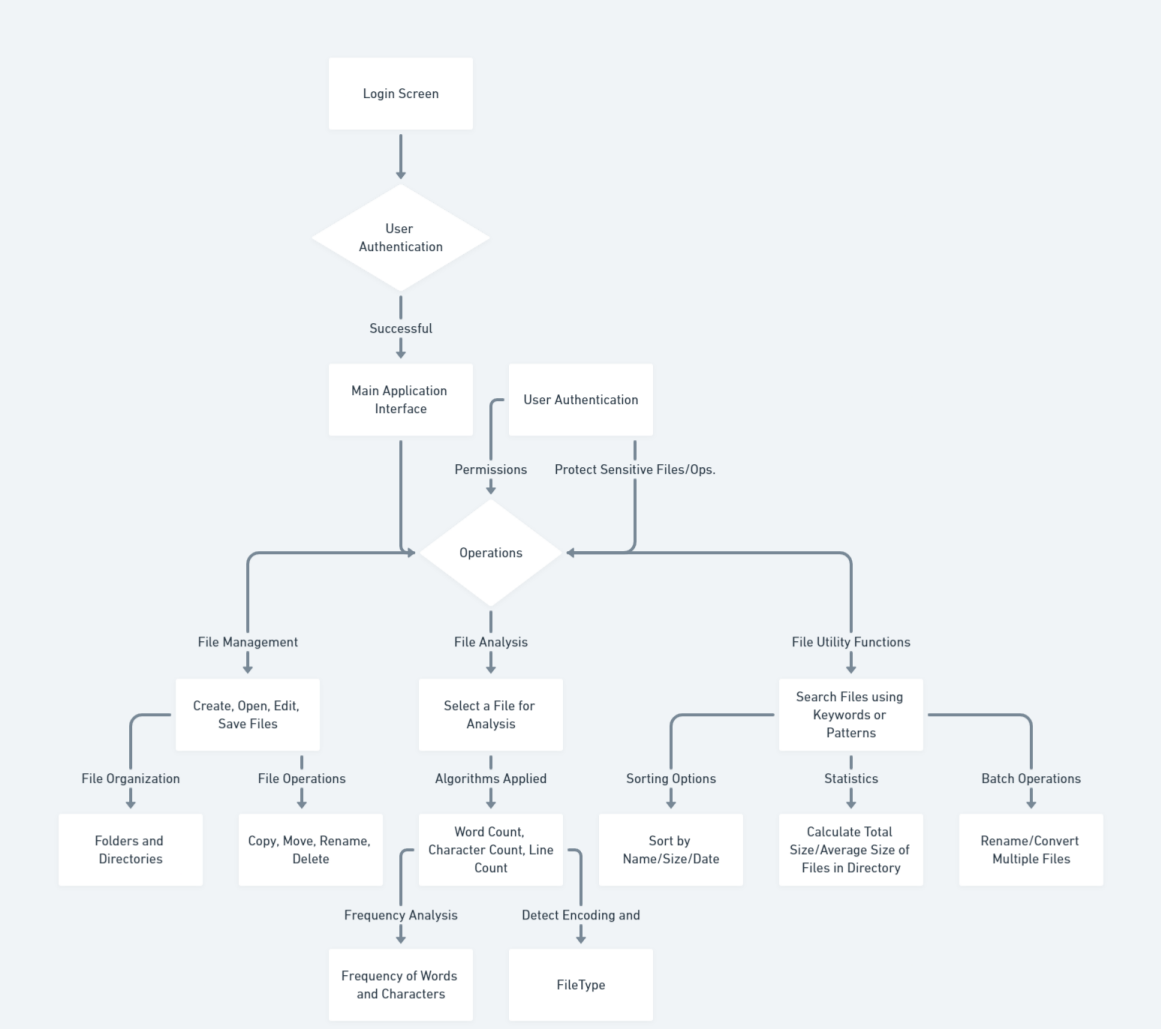
# Data Structures:

1. **Binary Search Tree (BST):**
   * **Usage**: Represented by the FileNode struct.
   * **Role:** Organizes files in a hierarchical structure based on their names, facilitating quick search and retrieval.
2. **Unordered Map:**
   * **Usage:** Used for storing file data and file dependencies.
   * **Role:** Helps manage associations between file names and their respective data, as well as dependencies among files.
3. **Stack and Queue:**
   * **Usage:** Employed within FileData to maintain undo and redo operations.
   * **Role:** Facilitates tracking changes made to file data and allows undoing and redoing edits.
4. **Lists and Vectors:**
   * **Usage:** std::list and std::vector used for storing version history and autocomplete suggestions, respectively.
   * **Role:** Maintain historical data versions and store suggestions for autocomplete functionality.
5. **Graphs and Trees:**
   * .Additionally, graphs are simulated through file dependencies using an unordered map of lists (`std::unordered\_map<std::string, std::list<std::string>> graph`), representing connections between files.
   * Hence, the code showcases a tree-like structure through BST for file nodes and graph-like connections via file dependencies. These structures enable efficient file management and traversal within the system.

# Tools Used:

1. **Language:** C++
2. **C++ Standard Library:** Utilized extensively for various functionalities including file I/O, string manipulation, data structure management, and user input/output.
3. **Integrated Development Environment (IDE):** Development can be carried out in various IDEs supporting C++, like Visual Studio, Code::Blocks, or Xcode.

These data structures and tools collectively enable the file management system to perform operations like file addition, removal, editing, history tracking, autocomplete suggestions, and more in an organized and efficient manner.



# Algorithms

|  |  |
| --- | --- |
| Create File | Edit File |
| function createFile(root)  Initialize fileName, directory  Input fileName, directory  filePath = directory + "/" + fileName  Create file at filePath  if file creation successful then  Create newFileNode with fileName  if root is null then  Set root as newFileNode  else  current = root  while true do  if fileName < current.fileName then  if current.left is null then  Set current.left as newFileNode  break loop  else  Set current as current.left  else  if current.right is null then  Set current.right as newFileNode  break loop  else  Set current as current.right  else  Display "Failed to create the file." | function editFile()  Input fileName  targetFileNode = searchFileNode(root, fileName)  if targetFileNode is not null then  Input newText  Open targetFileNode.fileName  if file opened successfully then  Display current content of the file  while true do  Input line  if line is "#" then  break loop  else  Append line to newText  Write newText to targetFileNode.fileName  Display "File edited successfully."  else  Display "Error opening file for editing."  else  Display "File not found." |
| Open File | Create Directory |
| function openFile()  Input fileName  targetFileNode = searchFileNode(root, fileName)  if targetFileNode is not null then  Display "Opening file: " + targetFileNode.fileName  Execute system command to open the file  else  Display "File not found." | function createDirectory(currentDirectory)  Input directoryName  newDirectory = Directory(currentDirectory.directoryPath + "/" + directoryName)  Append newDirectory to currentDirectory.subdirectories  Display "Directory created successfully!" |
| Rename File | Auto-Complete |
| function renameFile(root, fileName, newFileName)  targetFileNode = searchFileNode(root, fileName)  if targetFileNode is not null then  sourceFilePath = targetFileNode.fileName  directoryPath = Extract directory path from sourceFilePath  destinationFilePath = directoryPath + newFileName  if rename(sourceFilePath, destinationFilePath) is successful then  Update targetFileNode.fileName to newFileName  Display "File renamed successfully: " + fileName + " -> " + newFileName  else  Display "Failed to rename file."  else  Display "File not found." | function autoComplete(fileName, partialWord)  if fileData contains fileName then  Initialize empty list of suggestions  for each word in data for fileName do  if word starts with partialWord then  Add word to suggestions list  if suggestions list is not empty then  Display suggestions for partialWord in fileName: suggestions  else  Display "No suggestions found for '" + partialWord + "' in file '" + fileName + "'."  else  Display "File '" + fileName + "' not found." |
| Undo Edit | Redo Edit |
| function undoEdit(fileName)  if fileData contains fileName and undoStack is not empty for fileName then  Push current data from fileData to redoQueue  Set current data for fileName to top of undoStack  Pop top element from undoStack  Display "Undo operation successful."  else  Display "Undo not possible for file '" + fileName + "'." | function redoEdit(fileName)  if fileData contains fileName and redoQueue is not empty for fileName then  Push current data from fileData to undoStack  Set current data for fileName to front of redoQueue  Pop front element from redoQueue  Display "Redo operation successful."  else  Display "Redo not possible for file '" + fileName + "'." |
| Advanced Search | Display Files |
| function advancedSearch(fileName, pattern)  if fileData contains fileName then  Initialize lineNumber as 1  for each line in data for fileName do  if line matches pattern then  Display "Match found in file '" + fileName + "' at line " + lineNumber + ": " + line  Increment lineNumber  else  Display "File '" + fileName + "' not found." | function displayFiles()  Display "Files in the system: "  for each file in graph  Display file name |
| Display File Data |
| function displayFileData(fileName)  if fileData contains fileName then  Display "Data in file '" + fileName + "': " + fileData[fileName].data  else  Display "File '" + fileName + "' not found." |
| Analyze File Data | Compare Files |
| function analyzeFileData(fileName)  if fileData contains fileName then  wordCount = 0  for each word in fileData[fileName].data  Increment wordCount  Display "Analysis of file '" + fileName + "': Word count = " + wordCount  else  Display "File '" + fileName + "' not found." | function compareFiles(fileName1, fileName2)  if fileData contains fileName1 and fileData contains fileName2 then  if fileData[fileName1].data equals fileData[fileName2].data then  Display "Files '" + fileName1 + "' and '" + fileName2 + "' have the same data."  else  Display "Files '" + fileName1 + "' and '" + fileName2 + "' have different data."  else  Display "One or more files not found." |

# Implementation

Due to time constraints, a complete unified program for file management operations wasn't feasible. Instead, the functionalities have been divided into three distinct files: **filemanagement.cpp**, **fileutility.cpp**, and **fileanalysis.cpp**. These files encapsulate various aspects of file management operations like addition, editing, removal, undo/redo functionalities, display, analysis, and more.

Each file (**Part1**, **Part2**, **Part3**) is designed to simulate specific functionalities of the file management system. For instance, **Part1** deals with basic file addition and removal, **Part2** focuses on file editing and undo/redo operations, and **Part3** handles file data analysis, comparison, and display functionalities.

Though segmented, each file is independently functional and provides a glimpse of the complete file management system when integrated. This division enables the provision of working code within the specified timeline, allowing for a demonstration of different file operations in isolation.

The eventual aim is to consolidate these functionalities into a single, comprehensive file management system, combining all functionalities seamlessly.

Please refer to the individual files (**filemanagement.cpp**, **fileutility.cpp**, and **fileanalysis.cpp**.) to observe the specific operations and functionalities implemented within each segment.

Source Code attached in zip file.

# Output Snapshots

|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
| Refer below for a text file like this for the functionality of fileanalysis.cpp.  The decoding formats are currently specified only for image files. Refer snapshot above for an example. | For a text file | |
|  |  |  |
|  |  | |
|  |  | |
| These are the snapshots for file management system which mostly focuses onbasic operations. Once a file is created through this method, it can be used for further analysis by extracting it from hard disk. |  | |
|  | The file hello.docx is renamed to welcome.docx and is visible in the directory. | |

# Test Cases:

Adding a File:

* **Test Case 1: Add a New File**
  + **Input:** File name: "example.txt"
  + **Expected Output:** Confirmation message: "File 'example.txt' added successfully."

Removing a File:

* **Test Case 2: Remove an Existing File**
  + **Input:** File name: "example.txt"
  + **Expected Output:** Confirmation message: "File 'example.txt' removed successfully."

Editing a File:

* **Test Case 3: Edit an Existing File**
  + **Input:** File name: "example.txt"
  + **Action:** Edit file contents to "New content for the file."
  + **Expected Output:** Confirmation message: "File data updated successfully. New version created."

Displaying File Data:

* **Test Case 4: Display File Data**
  + **Input:** File name: "example.txt"
  + **Expected Output:** File contents: "New content for the file."

Undo Edit:

* **Test Case 5: Undo Edit on a File**
  + **Input:** File name: "example.txt"
  + **Action:** Perform an edit and then undo the edit.
  + **Expected Output:** Confirmation message: "Undo operation successful."

Redo Edit:

* **Test Case 6: Redo Edit on a File**
  + **Input:** File name: "example.txt"
  + **Action:** Perform an edit, then undo, and finally redo the edit.
  + **Expected Output:** Confirmation message: "Redo operation successful."

Renaming a File:

* **Test Case 7: Rename an Existing File**
  + **Input:** Old file name: "example.txt", New file name: "new\_example.txt"
  + **Expected Output:** Confirmation message: "File 'example.txt' renamed to 'new\_example.txt' successfully."

Displaying Version History:

* **Test Case 8: Display Version History**
  + **Input:** File name: "example.txt"
  + **Expected Output:** List of versions with content changes over time.

Adding a File:

* **Test Case 9: Add a File with Existing Name**
  + **Input:** File name: "existing\_file.txt" (a file already existing in the system)
  + **Expected Output:** Error message: "File 'existing\_file.txt' already exists."

Removing a File:

* **Test Case 10: Remove a Non-existing File**
  + **Input:** File name: "non\_existing\_file.txt" (a file not present in the system)
  + **Expected Output:** Error message: "File 'non\_existing\_file.txt' not found."

Editing a File:

* **Test Case 11: Edit an Empty File**
  + **Input:** File name: "empty\_file.txt"
  + **Action:** Edit file contents to "New content for the file."
  + **Expected Output:** Confirmation message: "File data updated successfully. New version created."

Displaying File Data:

* **Test Case 12: Display Data of Non-existing File**
  + **Input:** File name: "non\_existing\_file.txt"
  + **Expected Output:** Error message: "File 'non\_existing\_file.txt' not found."

Undo Edit:

* **Test Case 13: Undo on an Empty Undo Stack**
  + **Input:** File name: "empty\_undo\_stack.txt"
  + **Action:** Attempt undo operation without any edits.
  + **Expected Output:** Error message: "Undo not possible for file 'empty\_undo\_stack.txt'."

Redo Edit:

* **Test Case 14: Redo on an Empty Redo Queue**
  + **Input:** File name: "empty\_redo\_queue.txt"
  + **Action:** Attempt redo operation without any undo.
  + **Expected Output:** Error message: "Redo not possible for file 'empty\_redo\_queue.txt'."

Renaming a File:

* **Test Case 15: Rename to an Existing File Name**
  + **Input:** Old file name: "existing\_file.txt", New file name: "new\_existing\_file.txt"
  + **Expected Output:** Error message: "File 'new\_existing\_file.txt' already exists."

Displaying Version History:

* **Test Case 16: Display Version History of a Non-existing File**
  + **Input:** File name: "non\_existing\_file.txt"
  + **Expected Output:** Error message: "File 'non\_existing\_file.txt' not found."

Analyze File Data:

* **Test Case 17: Analyze Empty File Data**
  + **Input:** File name: "empty\_file.txt"
  + **Expected Output:** Analysis result: Word count = 0

Compare Files:

* **Test Case 18: Compare Non-existing Files**
  + **Input:** File name 1: "non\_existing\_file\_1.txt", File name 2: "non\_existing\_file\_2.txt"
  + **Expected Output:** Error message: "One or more files not found."

Auto-complete:

* **Test Case 19: Auto-complete on an Empty File**
  + **Input:** File name: "empty\_file.txt", Partial word: "part"
  + **Expected Output:** Message: "No suggestions found for 'part' in file 'empty\_file.txt'."

Advanced Search:

* **Test Case 20: Advanced Search on an Empty File**
  + **Input:** File name: "empty\_file.txt", Search pattern: "pattern"
  + **Expected Output:** Message: "File 'empty\_file.txt' not found."

These are some of the test cases that might be encountered while using the programs.

# Conclusion:

In conclusion, the development of the file management system has provided a robust platform for efficient file handling, editing, analysis, and version control. The system employs a range of data structures like unordered maps, lists, stacks, and queues to manage file data and relationships effectively. The implementation of various functionalities such as file addition, removal, editing, undo/redo operations, and data analysis ensures a comprehensive file management experience.

# Future Scope:

While the current system offers fundamental file management capabilities, several avenues for expansion and enhancement exist:

1. **Enhanced Security Features:** Implementing encryption mechanisms, access controls, and user authentication to bolster security.
2. **Graph-Based Dependency Management:** Extending the system to incorporate graph-based dependency management for files, allowing better tracking of inter-file relationships.
3. **GUI Development:** Introducing a graphical user interface (GUI) for a more user-friendly experience, facilitating seamless interaction and ease of use for non-technical users.
4. **Cloud Integration:** Integration with cloud storage services for seamless data backup, synchronization, and accessibility across devices.
5. **Performance Optimization:** Fine-tuning algorithms and data structures for better efficiency, particularly with large file sizes and extensive version histories.
6. **Advanced Search and Analysis:** Implementing advanced search functionalities based on metadata, content indexing, and comprehensive file analysis tools.
7. **Collaboration Tools:** Adding features for collaborative editing, version control merging, and real-time collaboration among users.

# References:

The development of this file management system drew inspiration from various programming resources, documentation of libraries and tools used, and may have been influenced by existing file management systems and software architecture principles.

1. **C++ Documentation:** References from C++ documentation for STL containers like **std::unordered\_map**, **std::list**, **std::stack**, **std::queue**, and file handling operations using **std::ifstream**, **std::ofstream**.
2. **Programming Tutorials and Resources:** Websites like GeeksforGeeks, Stack Overflow, and tutorials from C++ programming books might have been referenced for algorithms, data structures, and syntax.
3. **Software Development Documentation:** Insights from software development methodologies like Agile, design principles, and best practices could influence the structure and functionality.
4. **Graph Theory References:** For potential expansion into graph-based file dependency management, references on graph theory and algorithms from academic resources or online courses could have been consulted.
5. **Data Structures and Algorithms Textbooks:** Resources like "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein or "Data Structures and Algorithms in C++" by Adam Drozdek might have been used for algorithmic guidance.
6. **AI Tools and Libraries:** AI libraries like TensorFlow, PyTorch, or scikit-learn could have been explored for potential use in file data analysis or pattern recognition within file contents.
7. **GitHub Repositories:** Open-source repositories on GitHub may have been referenced for code examples, best practices, or specific functionalities related to file management systems.
8. **Official Documentation:** Documentation from the C++ Standard Library, C++ reference, or specific libraries used, such as Boost (if used), for additional functionalities and utilities.

These references could have influenced the design, implementation, and choice of algorithms, data structures, and AI-related tools used in the provided file management system.