Clarity, Structure, and Readability of the Code

Clarity and Structure:

- The code is well-structured with a clear separation of concerns. The `Directory` class encapsulates the logic related to directories and files, and the `FileSystem` class manages the interaction between the user and the file system.

Readability:

- Variable and function names are descriptive, contributing to code readability.

- Logical flow is clear, making it easy to understand the sequence of operations.

Docs – Implementation Details, Design Decisions, etc.

Implementation Details:

- The code implements a basic in-memory file system with functionalities like creating directories, navigating directories, listing contents, creating files, and deleting files.

- The implementation uses maps to store directories and files, and each directory keeps track of its parent directory.

Design Decisions:

- The use of the `Directory` class provides a clear structure for organizing directories and files.

- The use of `map` data structures allows for efficient lookup operations, maintaining a balance between simplicity and performance.

Instructions on How to Run and Test the Program:

==================================================

1. Compile the Code:

- Use a C++ compiler (e.g., g++) to compile the code. For example:

```bash

G++ -o filesystem filesystem.cpp

```

2. Run the Executable:

- Execute the compiled program:

```bash

./filesystem

```

3.Interact with the File System:

- Enter commands at the prompt (e.g., `mkdir dir1`, `ls`, `cd dir1`, etc.).

4. Exit the Program:

- To exit the program, type `exit`.

Improvements Made to the Design:

================================

1. Memory Cleanup:

- Added a destructor to the `Directory` class to free allocated memory.

2.Error Handling:

- Improved error handling for user input, providing more informative messages.

3. Input Validation:

- Checked for missing arguments before processing commands to enhance user experience.

4. Dynamic Memory Management:

- Suggested using smart pointers for more efficient memory management.

5. Bonus: Save and Reload State:

- Suggested adding functions to save and load the state of the file system.

Summary:

The provided C++ code serves as a good foundation for an in-memory file system. It demonstrates efficient time complexities for critical operations, maintains clarity and readability, and incorporates suggestions for improvement in terms of error handling, input validation, and dynamic memory management.