In memory file system

Key Points:

- Implementation done using c++.
- **Classes**: Two main classes are used: `File` and `Directory`.
    - `File`: Represents a file with attributes like `name` and `content`.
    - `Directory`: Represents a directory containing files (`unordered_map<string, File*>`) and subdirectories (`unordered_map<string, Directory*>`).
  FileSystem Class:
    - **Private Members**:
        - `root`: Represents the root directory.
        - `currentDir`: Points to the current working directory.
        - `prevDirs`: A stack to maintain the history of directories navigated.
    - **Public Member Functions**:
        - `mkdir`: Creates new directories by parsing the given path.
        - `cd`: Changes the current directory based on the provided path.
        - `ls`: Lists the contents (directories and files) of the current directory.
        - `touch`: Creates a new file in the specified directory.
        - `echo`: Adds content to a file.
        - `cat`: Displays the content of a file.
        - `mv`: Moves a file or directory from a source to a destination.
        - `cp`: Copies a file or directory from a source to a destination.
        - `rm`: Removes a file or directory.
        - `grep`: Searches for a pattern in a file.

`mkdir(string path)`

- **Data Structure**:
    - `unordered_map<string, Directory*> dirs` within the `Directory` class.
- **Usage**:
    - Creates new directories by parsing the given `path`.
    - Uses `unordered_map` to efficiently store and access subdirectories in the current directory.

`cd(string path)`

- **Data Structure**:
    - `unordered_map<string, Directory*> dirs` within the `Directory` class.
    - `stack<Directory*> prevDirs` within the `FileSystem` class.

- **Usage**:
  - Navigates directories based on the provided `path`.
  - `unordered_map` assists in efficiently locating directories.
  - `stack` maintains a history of visited directories for easier navigation back using `..`.

`ls()`

- **Data Structure**:
  - `unordered_map<string, File*> files` and `unordered_map<string, Directory*> dirs` within the `Directory` class.
- **Usage**:
  - Lists the contents of the current directory.
  - Utilizes `unordered_map` to display directories and files efficiently.

`touch(string path)`

- **Data Structure**:
  - `unordered_map<string, Directory*> dirs` within the `Directory` class.
  - `unordered_map<string, File*> files` within the `Directory` class.
- **Usage**:
  - Creates a new file in the specified directory.
  - Uses `unordered_map` to check for existing directories/files and to add a new file if it doesn't exist.

`echo(string text, string filename)`

- **Data Structure**:
  - `unordered_map<string, File*> files` within the `Directory` class.
- **Usage**:
  - Adds content to an existing file or creates a new file if it doesn't exist.
  - Utilizes `unordered_map` to efficiently store and access files within the directory.

`cat(string filename)`

- **Data Structure**:
  - `unordered_map<string, File*> files` within the `Directory` class.
- **Usage**:
  - Displays the content of a file.
  - Uses `unordered_map` to find and access the file efficiently.

`mv(string sourcePath, string destinationPath)`

- **Data Structure**:
  - `unordered_map<string, Directory*> dirs` and `unordered_map<string, File*> files` within the `Directory` class.
- **Usage**:
  - Moves a file or directory from a source to a destination.
  - Utilizes `unordered_map` to locate source and destination directories/files for the move operation.

`cp(string sourcePath, string destinationPath)`

- **Data Structure**:
  - `unordered_map<string, Directory*> dirs` and `unordered_map<string, File*> files` within the `Directory` class.
- **Usage**:
  - Copies a file or directory from a source to a destination.
  - Uses `unordered_map` to locate source and destination directories/files for the copy operation.

`rm(string path)`

- **Data Structure**:
  - `unordered_map<string, Directory*> dirs` and `unordered_map<string, File*> files` within the `Directory` class.
- **Usage**:
  - Removes a file or directory.
  - Utilizes `unordered_map` to locate and delete files or directories within the specified path.

`grep(const string& pattern, const string& filename)`

- **Data Structure**:
  - `unordered_map<string, File*> files` within the `Directory` class.
- **Usage**:
  - Searches for a pattern in a file's content.
  - Uses `unordered_map` to access and search through the content of the specified file.
-
  -

Design Decisions:

- **Directory Navigation**:
  - Implemented `cd` functionality to navigate through directories.
  - Maintained a stack `prevDirs` to store the history of visited directories for easy navigation back using `..`.
  - Implemented error handling for invalid paths or directory navigation.
- **File and Directory Operations**:
  - Operations like `touch`, `echo`, `cat`, `mv`, `cp`, `rm`, `grep` are implemented to mimic file system behavior.
  - Each operation involves parsing the provided paths, checking for existing files/directories, and performing the required action.
- **Input Handling**:
  - Utilized `cin` and `getline` for taking user inputs and parsing commands/paths.

Instruction

Running the Program:

1. **Setup Environment:**
   - Go to chrome browser.
   - Navigate to link https://replit.com/~
   - Click on Create Repl in c++ and paste the provide main.cpp code in github.
   - If above does not work simply open the below link in browser
   - https://replit.com/@PriyankaMahara1/Project-Inito#main.cpp
2. **Run the Program:**
   - Run the program.
   - the file system program, prompting you to enter commands.

Testing the Program:

Here are some test cases you can use to check the functionality of the file system:

1. **Creating Directories:**
   - Type `mkdir` command to create directories and hit enter:

     `Home1`
     `home/user/documents`
2. **Changing Directories:**
   - Use the `cd` command to change directories and hit enter:

     `home/user`
     `..`
3. **Listing Contents:**
   - Use the `ls` command to list the contents of directories and hit enter:
4. **Creating Files:**
   - Use the `touch` command to create files and hit enter:

     `home/user/documents/file1.txt`
     `home/user/documents/file2.txt`
5. **Adding Content to Files:**
   - Use the `echo` command to add content to a file and hit enter:

     `Hello World!` (hit enter 2 times)
     `home/user/documents/file1.txt`
6. **Viewing File Contents:**
   - Use the `cat` command to view file contents and hit enter:

     `home/user/documents/file1.txt`
7. **Searching for Patterns in Files:**
   - Use the `grep` command to search for patterns in file contents and hit enter:

     `World` (hit enter 2 times)
     `home/user/documents/file1.txt`
8. **Moving Files:**
   - Use the `mv` command to move files to different directories and hit enter:

     `home/user/documents/file1.txt (source)`

```
home/user(destination)
```

9. **Copying Files:**
    - Use the **`cp`** command to copy files to different directories and hit enter:

```
home/user/documents/file2.txt(source)
home/user(destination)
```

10. **Removing Files/Directories:**
    - Use the **`rm`** command to remove files or directories and hit enter:

```
file2.txt
```

11. **Exit the Program:**
    - Use the **`exit`** command to quit the file system program and hit enter:

```
exit
```