**DAY- 2**

**Objective: Develop a console-based file explorer application in C++ that interfaces with the Linux operating system to manage files and directories.**

**PSEUDOCODE**

Start

// Step 1: Initialization

Initialize application

Setup environment and resources

// Step 2: User Interaction

Display available operations (Move file, Copy file, Delete file, Create file, Search file, Manage file permissions)

Get user input for desired operation

// Step 3: Command Processor

IF user selects "Move file"

Perform Move file operation

ELSE IF user selects "Copy file"

Perform Copy file operation

ELSE IF user selects "Delete file"

Perform Delete file operation

ELSE IF user selects "Create file"

Perform Create file operation

ELSE IF user selects "Search file"

// Step 4a: File Search

Get search criteria (e.g., file name or pattern)

Search for files matching the criteria in the specified directory

Display search results to the user

ELSE IF user selects "Manage file permissions"

// Step 4b: File Permission Management

Get file or directory name for which to manage permissions

Get desired permission changes from user

Modify file or directory permissions accordingly

Display permission modification results to the user

ELSE

Display invalid operation message

// Step 5: Functional Execution

Perform the selected file operation (if it is not a search or permission management)

// Step 6: Logger

Log the operation details (e.g., success, failure, operation type)

// Step 7: Operation Successful?

IF operation was successful

Continue normal execution

ELSE

// Step 8: Error Handling

Handle error

Display error message to the user

// Step 9: Exit application

End

Explanation:

**Start:**

* The program begins execution.

**Step 1: Initialization:**

* The application is initialized, which includes setting up the necessary environment and resources. This might involve configuring paths, loading configurations, or initializing any required libraries.

**Step 2: User Interaction:**

* The program displays a list of available file operations to the user.
* The user selects an operation (e.g., move, copy, delete, create, search, or manage permissions).
* The program captures the user’s choice for further processing.

**Step 3: Command Processor:**

* The program processes the user's input and directs the flow based on the selected operation:
  + **Move File:** Executes the file move operation.
  + **Copy File:** Executes the file copy operation.
  + **Delete File:** Executes the file delete operation.
  + **Create File:** Executes the file creation operation.
  + **Search File:**
    - Gathers search criteria from the user.
    - Searches for files matching the criteria in the specified directory.
    - Displays the search results.
  + **Manage File Permissions:**
    - Asks for the file or directory name for which permissions need to be managed.
    - Collects the desired permission changes from the user.
    - Modifies the file or directory permissions accordingly.
    - Displays the results of the permission modification.
* If an invalid option is selected, the program informs the user and waits for valid input.

**Step 5: Functional Execution (if applicable):**

* If the selected operation involves moving, copying, deleting, or creating a file, the program executes that function. For search and permission management, the specific steps outlined in the command processor handle execution.

**Step 6: Logger:**

* The program logs the details of the operation, such as whether it was successful or failed, and the type of operation. This is useful for debugging, auditing, or tracking usage.

**Step 7: Operation Successful?**

* The program checks if the operation was successful:
  + If successful, the program will be end
  + If unsuccessful, it triggers error handling and goes to use interaction.

**Step 8: Error Handling:**

* If an error occurs during any of the operations, the program handles the error by:
  + Logging the error details.
  + Displaying an error message to the user to inform them of the issue.

**Step 9: Exit:**

* Finally, the program terminates.