

Muhammad Suleman Faisal

# User Guide

(to a file management system)

## Introduction:

When the user runs the code along with the given sample data file, the user is given a menu of commands. Like this:

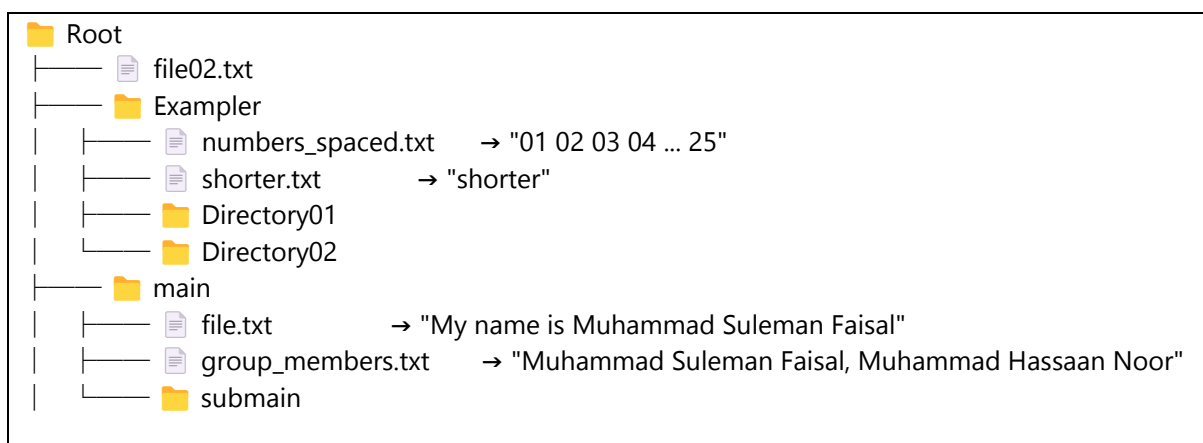
```
Commands:
01. create <filename>
02. delete <filename>
03. mkdir <dirname>
04. chdir <dirname>
05. move <source> <target>
06. open <filename>
07. close <filename>
08. write <filename> <text>
09. write_at <filename> <pos> <text>
10. read <filename>
11. read_from <filename> <start> <size>
12. move_within <filename> <start> <size> <target>
13. truncate <filename> <size>
14. memory_map
15. help
16. ls
17. exit
> /*write commands here*/
```

User can write these commands in the cli (a while loop) until he/she writes “exit” command.

The initial given Directory Structure and a brief description of commands with examples is given below.

## Current Directory Structure:

The structure of whole data store in sample.dat is given as following:



The function inside the FileSystem, memory\_map shows the same tree as:

```
> memory_map
[DIR] Exempler
  [DIR] Directory01
  [DIR] Directory02
  [FILE] numbers_spaced.txt
  [FILE] shorter.txt
[DIR] main
  [DIR] submain
  [FILE] file.txt
  [FILE] group_members.txt
[FILE] file02.txt
```

## Functions Implemented:

### 1. create <file\_name>

This takes file\_name as input and creates a file of given name if there is no file of such name in current directory. Example use:

```
Exempler> create file01.txt
File created: file01.txt
Exempler> ls
Exempler>
Contents of directory 'Exempler':
[FILE] file01.txt
Exempler> █
```

If file already exists:

```
Exempler> create file01.txt
File already exists.
```

### 2. delete <file\_name>

This takes file\_name as input and deletes file of such name if it is present in the current directory. Example use:

```
Exempler> delete file01.txt
File deleted: file01.txt
Exempler> ls
Exempler>
Contents of directory 'Exempler':
Exempler> █
```

If file's name is misspelled:

```
Exempler> delete dille01.cpp
File not found.
Exempler> █
```

### 3. mkdir <directory\_name>

This takes directory\_name as input and makes a directory of given name in the current directory if a directory of this name is not already present in the current directory.

Example use:

```

Exampler> ls
Exampler>
Contents of directory 'Exampler':
[DIR] Directory01
Exampler> mkdir Directory02
Directory created: Directory02
Exampler> ls
Exampler>
Contents of directory 'Exampler':
[DIR] Directory01
[DIR] Directory02
Exampler>

```

If directory of such name already exists:

```

Exampler> ls
Exampler>
Contents of directory 'Exampler':
[DIR] Directory01
Exampler> mkdir Directory01
Directory already exists.

```

If the name of directory is root (the code does not allow making 'root' directory, so that it does not confuse with main 'root'. It is just a preventive measure):

```

Exampler> mkdir root
Cannot create another 'root' directory.
Exampler>

```

#### 4. `chdir <directory_name>`

Takes `directory_name` as input and **changes current directory** to directory of the given name if such directory exists inside the current directory **OR** changes current directory to parent directory of current directory if `directory_name` is '..' (Two dots). Example use:

```

Exampler> chdir Directory01
Exampler>Directory01>

```

*(Note: Current directory can be seen from path before each command like*

*Example>Directory01>)*

If directory is not in current directory:

```

Exampler> chdir none
Directory not found.
Exampler>

```

If user wants to go back to the parent directory, (`directory_name` is ".."):

```

Exampler>Directory01> chdir ..
Exampler>

```

#### 5. `ls`

This command lists all files and directories inside current directory and also mentions name of the current directory. Example use:

```
Exampler> ls

Contents of directory 'Exampler':
  Directory01
  Directory02
  numbers_spaced.txt
  shorter.txt
Exampler>
```

#### 6. move <source> <target>

This command moves the content of given source file in the current directory to a new file, with a target name. If target exists, then it moves contents into that file, else if target does not yet exist it makes a new file with name target and moves content of the source file into that target. Example use:

```
Exampler> create source.txt
File created: source.txt
Exampler> write source.txt Contents of source file
Exampler> move source.txt target.txt
Moved file: source.txt -> target.txt
Exampler> read target.txt
Contents of source file
Exampler> read source.txt
File not found.
Exampler> ls
Exampler>
Contents of directory 'Exampler':
[DIR] Directory01
[DIR] Directory02
[FILE] target.txt
Exampler>
```

*(Note: As can be seen from the above example that after moving the contents the relevant module deletes the source file.)*

#### 7. open <file\_name>

This command opens file of given file\_name if it is present in current directory. Example use:

```
Exampler> ls
Exampler>
Contents of directory 'Exampler':
[DIR] Directory01
[DIR] Directory02
[FILE] toOpen.txt
Exampler> open toOpen.txt
Exampler>
```

If file does not exist in the current directory:

```

Exampler> ls
Exampler>
Contents of directory 'Exampler':
[FILE] file01.txt
Exampler> open file02.txt
File not found.
Exampler>

```

Although this function is provided separately, main purpose of this is as a helping function inside. For example, this is used to run commands like,

```

write <file_name> <text>,
write_at <file_name> <pos> <text>,
read <file_name>,
read_from <file_name> <start> <size>,
move_within <file_name> <start> <size> <target>,
truncate <file_name> <size>.

```

Improvement suggestion: We have added 'openFile' command hardcoded in main() for now, and it may be removed if we want to restrict user from using these six above functions after opening the file. As for now, open and close are kind of useless in sense that user does not need them. But these will be implemented in next lab where multiple users use this one app code.

*(Note: The main module used in all these above functions is openFile (const string& filename))*

#### 8. `close <file_name>`

This command is used to close an open file. Example use:

```

Exampler> open toOpen.txt
Exampler> close toOpen.txt
File closed.
Exampler>

```

If specified file in current directory is not open, it closes it even then. That is because closeFile(filename) module works by setting is\_open boolean to false as a snippet is given below:

```

currentDir->files[filename].is_open = false; // Mark the file as closed
cout << "File closed.\n";

```

#### 9. `write <file_name> <text>`

This command takes file\_name and text as input. It opens specified file\_name and write text inside this file. Example use:

```

Exampler> write toWrite.txt This is text written in file when file is empty initially
Exampler> read toWrite.txt
This is open fileThis is text written in file when file is empty initially
Exampler>

```

Writing in file in which something is already written:

```

Exampler> write toWrite.txt Writing where something is already written.
Exampler> read toWrite.txt
This is open fileThis is text written in file when file is empty initiallyWriting where something is already written.
Exampler>

```

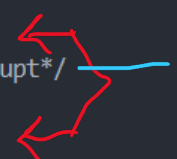
10. `write_at <file_name> <pos> <text>`

This commands overwrites on currently present file of name `file_name` starting from position `pos` and writing text. Example use:

```

Exampler> create written_at.txt
File created: written_at.txt
Exampler> write written_at.txt The text that was initially written in a file
Exampler> read written_at.txt
The text that was initially written in a file
Exampler> write_at written_at.txt 10 /*Interrupt*/
Exampler> read written_at.txt
The text t/*Interrupt*/ally written in a file
Exampler>

```



When initial file is empty:

```

Exampler> create written_at.txt
File created: written_at.txt
Exampler> write_at written_at 25 So Far Away
File not found.
Exampler> read written_at
File not found.
Exampler> write_at written_at.txt 25 So Far Away
Exampler> read written_at.txt
So Far Away
Exampler>

```

When initial file is shorter than pos given:

```

Exampler> create written_at.txt
File created: written_at.txt
Exampler> write written_at.txt Shorter
Exampler> write_at written_at.txt 25 So Far Away
Exampler> read written_at.txt
Shorter
So Far Away
Exampler>

```

11. `read <file_name>`

This command opens file with `file_name` from current directory and prints content of it on command line if there is any. Example use:

```

Exampler> create read_file.txt
File created: read_file.txt
Exampler> write read_file.txt Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit es
se cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Exampler> read read_file.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nul
la pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Exampler>

```

## 12. read\_from <file\_name> <start> <size>

This reads from start position till size, inside of a file with name file\_name inside the current directory after opening it. Example use:

```

Exampler> write numbers_spaced.txt 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Exampler> read_from numbers_spaced.txt 10 10
4 05 06 07
Exampler> read_from numbers_spaced.txt 20 23
08 09 10 11 12 13 14 1
Exampler>

```

If given pos is greater than content of file\_name:

```

Exampler> create shorter.txt
File created: shorter.txt
Exampler> write shorter.txt shorter
Exampler> read_from shorter.txt 20 5

Exampler>

```

If sum of given pos and size extends beyond contents of the given file:

```

Exampler> read_from numbers_spaced.txt 20 1000
Warning: Requested size exceeds file content. Truncating read.
08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Exampler>

```

## 13. move\_within <file\_name> <start> <size> <target>

This command moves content of a file with name file\_name of size size, from position start to position target. Example use:

```

Exampler> read numbers_spaced.txt
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Exampler> move_within numbers_spaced.txt 10 20 25
Exampler> read numbers_spaced.txt
01 02 03 011 12 13 14 15 4 05 06 07 08 09 10 16 17 18 19 20 21 22 23 24 25
Exampler>

```

If target location is greater than the size of the file:

```

Exampler> read numbers_spaced.txt
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Exampler> move_within numbers_spaced.txt 20 25 100
Error: Target position out of bounds.
Exampler>

```

If sum of given start and size is invalid:



```
Exampler> read numbers_spaced.txt
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Exampler> move_within numbers_spaced.txt 25 50 10
Error: Start position or size out of bounds.
Exampler> 
```

#### 14. truncate <file\_name> <size>

This reduces the size file to given size, by truncating all content of file file\_name within current directory which is above size. Example use:

```
Exampler> read numbers_spaced.txt
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Exampler> truncate numbers_spaced.txt 35
Exampler> read numbers_spaced.txt
01 02 03 04 05 06 07 08 09 10 11 12
Exampler> 
```

If size is larger than that of the contents of the file:

```
Exampler> read shorter.txt
shorter
Exampler> truncate shorter.txt 20
Warning: Size exceeds current content. No truncation performed.
Exampler> 
```

#### 15. memory\_map

This command shows structure of Directories and file when it is called. This function works by calling showMemoryMap(dname) recursively on all directories and prints files on each recursion level starting from root directory. There is a main root directory, and everything is within that directory. Example use:

```
> memory_map
├── Exempler
│   ├── Directory01
│   ├── Directory02
│   ├── numbers_spaced.txt
│   └── shorter.txt
├── main
│   ├── submain
│   ├── file.txt
│   ├── group_members.txt
│   └── file02.txt
> 
```

This means that there are three direct children of root, then [DIR] Exempler has four children and [DIR] main has three children.

*(Note: memory\_map can be used in any directory, but it shows overall structure taking from .dat file because of its implementation in main(...){...} function)*

#### 16. help OR help <command>

Whenever this command is called, it shows all commands and their brief description like use. Example use:

```
Exampler> help
Available Commands:

01. create <filename>          - Create a new file in current directory
02. delete <filename>          - Delete a file from current directory
03. mkdir <dirname>            - Create a new directory
04. chdir <dirname>            - Change to specified directory (use '..' to go up)
05. ls                          - Lists all files and directories in the current directory
06. move <source> <target>     - Rename/move a file
07. open <filename>            - Open a file for writing
08. close <filename>           - Close an opened file
09. write <filename> <text>    - Write text at the end of file
10. write_at <filename> <pos> <text> - Write text at specific position
11. read <filename>            - Read entire file content
12. read_from <filename> <start> <size> - Read part of file
13. move_within <filename> <start> <size> <target> - Move internal file data
14. truncate <filename> <size> - Cut file size to specified length
15. memory_map                 - Show current directory and files tree
16. help                       - To show work of available commands
17. exit                       - Exit the program
Exampler> █
```

And if user write help <command>, this tells one line help about that command. Example use:

```
> help exit
17. exit          - Exit the program
> help help
16. help          - To show work of available commands
```

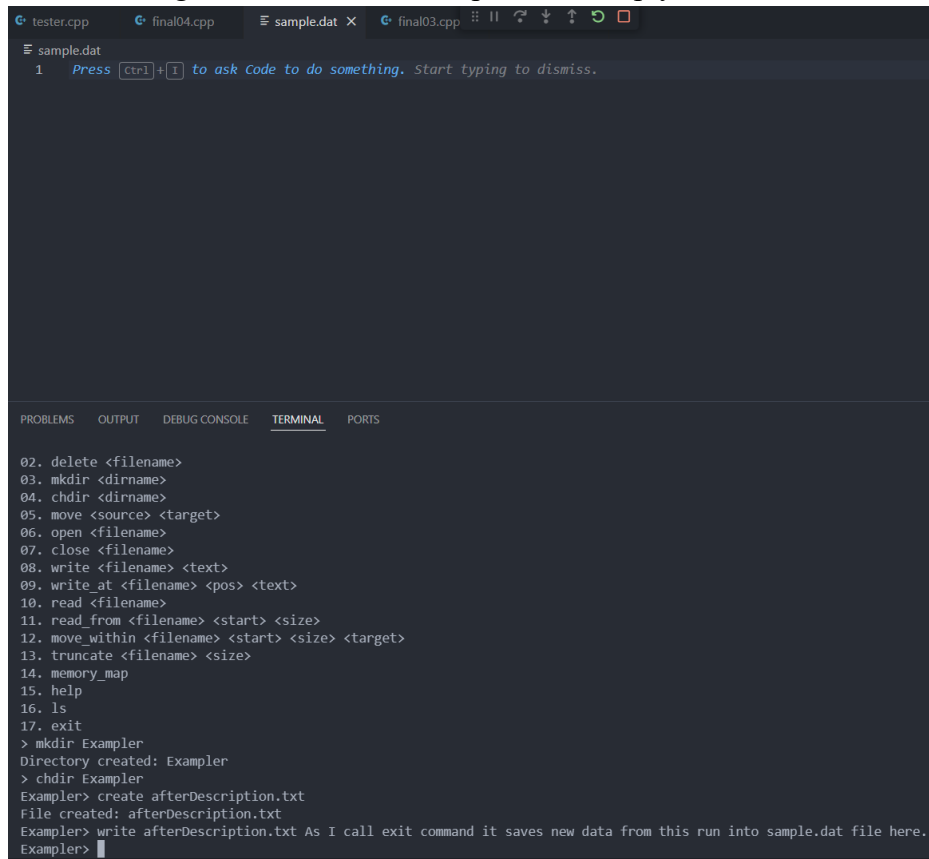
If command is invalid:

```
> help fun
Unknown command. Use 'help' to see the list of available commands.
> █
```

17. exit

This command before exiting the while loop, saves all data from current run into sample.dat file. Example use:

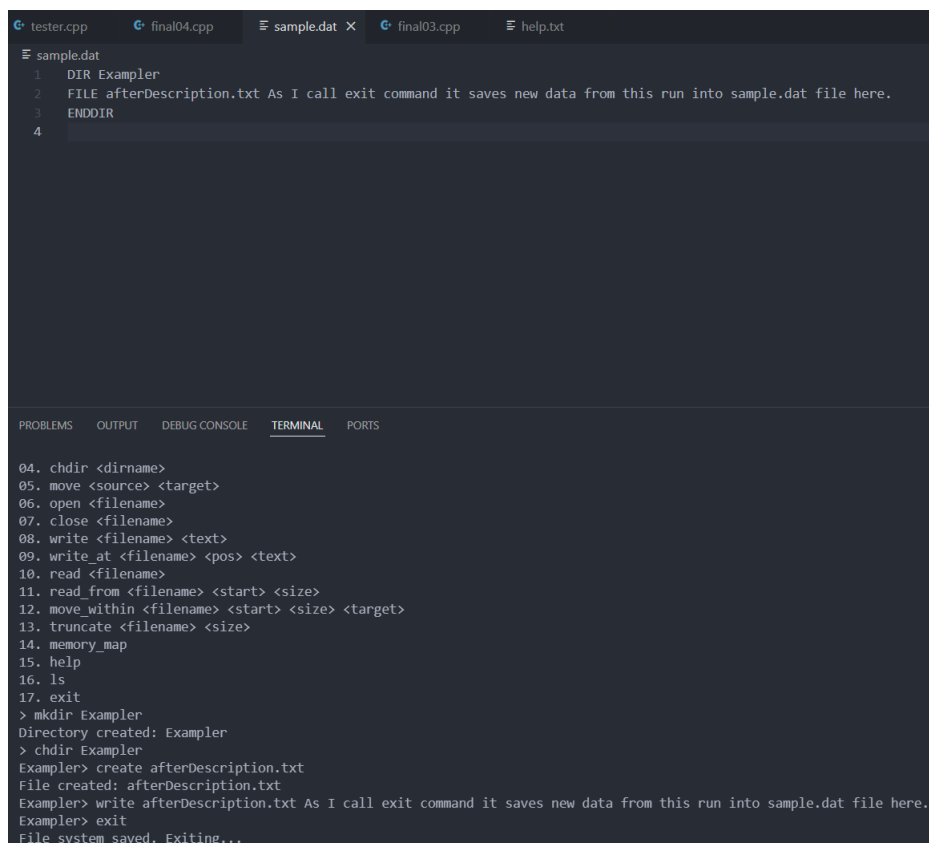
Before calling exit, for instance sample.dat is empty, like this:



The screenshot shows a code editor with a tab for 'sample.dat'. The file content is a single line: `1 Press [Ctrl]+I to ask Code to do something. Start typing to dismiss.`. Below the editor is a terminal window with the following commands and output:

```
02. delete <filename>
03. mkdir <dirname>
04. chdir <dirname>
05. move <source> <target>
06. open <filename>
07. close <filename>
08. write <filename> <text>
09. write_at <filename> <pos> <text>
10. read <filename>
11. read_from <filename> <start> <size>
12. move_within <filename> <start> <size> <target>
13. truncate <filename> <size>
14. memory_map
15. help
16. ls
17. exit
> mkdir Exemplar
Directory created: Exemplar
> chdir Exemplar
Exemplar> create afterDescription.txt
File created: afterDescription.txt
Exemplar> write afterDescription.txt As I call exit command it saves new data from this run into sample.dat file here.
Exemplar>
```

When exit is called:



The screenshot shows the same code editor with 'sample.dat' now containing four lines of text:

```
1 DIR Exemplar
2 FILE afterDescription.txt As I call exit command it saves new data from this run into sample.dat file here.
3 ENDDIR
4
```

The terminal window below shows the execution of the 'exit' command:

```
04. chdir <dirname>
05. move <source> <target>
06. open <filename>
07. close <filename>
08. write <filename> <text>
09. write_at <filename> <pos> <text>
10. read <filename>
11. read_from <filename> <start> <size>
12. move_within <filename> <start> <size> <target>
13. truncate <filename> <size>
14. memory_map
15. help
16. ls
17. exit
> mkdir Exemplar
Directory created: Exemplar
> chdir Exemplar
Exemplar> create afterDescription.txt
File created: afterDescription.txt
Exemplar> write afterDescription.txt As I call exit command it saves new data from this run into sample.dat file here.
Exemplar> exit
File system saved. Exiting...
```

## Some Additional Features:

### 18. suggestCommand(command)

Whenever user writes commands which does not match any of the previous 17 commands, that command input goes into suggestCommand(command) function which suggests nearest command by calculating levenshtein distance of given command and available 17 commands, if distance comes out to be less than 3, or says that “Invalid command” if distance is greater than that of 3. Example use:

```
Did you mean: 'ls'?  
> fast  
Did you mean: 'ls'?  
> invalid  
Unknown command. No similar command found.
```

As it can be seen, it is far from perfect but better than nothing.

### 19. displayPath ()

This is an original function of the FileSystem. This is run at start of each while loop, so just before getting input command from the user. This works by maintaining path inside a stack structure.