COMMAND LINE INTERPRETER

In this task, the objective is to develop a Command Line Interpreter (CLI) for the operating system.

The CLI must enable users to input commands via the keyboard. Following the user's input and pressing enter, the string should be parsed, and the specified command executed.

The CLI will consistently accept various commands from the user until the user inputs "exit," at which point the CLI will terminate. The program should be structured around two main classes: **Parser and Terminal.**

Program Structure:

Parser Class:

- Responsible for parsing the user's input and extracting relevant information.
- Should have methods for interpreting different commands.

Terminal Class:

- Manages the overall functioning of the CLI.
- Accepts user input, communicates with the Parser to interpret commands, and executes the corresponding actions.
- Continues accepting commands until the user inputs "exit."

Required Commands:

Command Name	***What You Must Implement***
echo	Takes **1 argument** and prints it.
pwd	Takes **no arguments** and prints the current path.
cd	Implement all these cases: 1. **cd** takes **no arguments** and changes the current path to the path of your home directory. 2. **cd** takes **1 argument** which is “..” (e.g. **cd ..**) and changes the current directory to the previous directory. 3. **cd** takes **1 argument** which is either the **full path or the relative (short) path** and changes the current path to that path.
ls	Takes **no arguments** and lists the contents of the current directory sorted alphabetically.
ls -r	Takes **no arguments** and lists the contents of the current directory in **reverse order.**
mkdir	Takes **1 or more arguments** and creates a directory for each argument. Each argument can be: • **Directory name** (in this case the new directory is created in the current directory) **Path** (full/short) that ends with a directory name (in this case the new directory is created in the given path)
rmdir	Implement all these cases: 1. **rmdir** takes **1 argument** which is “*” (e.g. **rmdir ***) and removes all the **empty directories** in the current directory. • **rmdir** takes **1 argument** which is either the **full path or the relative (short) path** and removes the given directory **only if it is empty.**

touch	2. Takes **1 argument** which is either the **full path or the relative (short) path** that ends with a file name and creates this file.
cp	Takes **2 arguments**, both are files and copies the first onto the second.
cp -r	Takes **2 arguments**, both are directories (empty or not) and copies the first directory (with all its content) into the second one.
rm	Takes **1 argument** which is a file name that exists in the current directory and removes this file.
cat	Takes **1 argument** and prints the file's content or takes **2 arguments** and concatenates the content of the 2 files and prints it.
wc	Wc stands for "word count," and as the name suggests, it is mainly used for counting purpose. By default, it displays **four-columnar output.** First column shows **number of lines** present in a file specified, second column shows **number of words** present in the file, third column shows **number of characters** present in file and fourth column itself is the **file name** which are given as argument Example: `wc file.txt` Output: `9 79 483 file.txt` Explanation: `# 9 lines, 79 word, 483 character with spaces, file name`
>	Format: ***command > FileName*** Redirects the output of the first command to be written to a file. If the file doesn't exist, it will be created. If the file exists, its original content will be **replaced.** Example:

	`echo Hello World >`
	`myfile.txt ls >`
	`file`

>>	Like > but **appends** to the file if it exists.

history	Takes no parameters and displays an enumerated list with the commands you've used in the past Example:
	`history`
	Output:
	`1  ls`
	`2  mkdir tutorial`
	`3  history`