PRACTICE PROBLEMS

Problem 1: Implement wc (word count) using file descriptors

Write a program that takes a filename as input and prints:

- Number of lines
- Number of words
- Number of characters

using only **open, read, and close** system calls (no ifstream, no fopen).

Problem 2: Implement tee command

Write a program that reads from **STDIN** and writes to both **STDOUT** and a file.

- File name should be passed as a command-line argument.
- Use only system calls (**`read`**, **`write`**, **`open`**).

Problem 3: Simple pipeline (ls | wc -l)

Write a program that mimics ls | wc -l using **pipe, fork, and dup2.**

- The first child runs ls.
- The second child runs wc -l.
- Parent just waits for both

Problem 4: Two-Process Text Filter

Write a program that:

1. Creates **two child processes** connected by a pipe.
2. The **first child** reads from an input file (`input.txt`), converts all characters to **uppercase**, and writes the result into the pipe.
3. The **second child** reads from the pipe, replaces all spaces `' '` with underscores `'_'`, and writes the result to an output file (`output.txt`).
4. The **parent** only sets up the pipe, forks the two children, and waits for them to finish.

Problem 5: Mini Command Runner

Write a program that acts like a very **basic shell**, but only supports **one feature at a time**.

Your shell should:

1. Execute a single command entered by the user (like ls, pwd, echo hello).
2. If the user enters multiple commands separated by ;, execute them sequentially one after the other.

   Example:

   ls ; pwd ; echo "done"

   - should run ls, then pwd, then echo "done".
3. If the user enters exit, the program should terminate.
4. On pressing Ctrl+C, the shell should simply print a message "Type 'exit' to quit" instead of killing itself.