1. Test File (testfile.txt)

Create a file named testfile.txt with the following content:
Hello world
This is a test
another test line
HELLO AGAIN
Don't match this line
Testing one two three

2. Make the Script Executable

In your terminal, run:
chmod +x mygrep.sh

3. Hands-On Validation (Expected Output)

Run these commands in your terminal. You should see the following output (capture this in your screenshot):
$ ./mygrep.sh hello testfile.txt
Hello world
HELLO AGAIN

$ ./mygrep.sh -n hello testfile.txt
1:Hello world
4:HELLO AGAIN

$ ./mygrep.sh -vn hello testfile.txt
2:This is a test
3:another test line
5:Don't match this line
6:Testing one two three

$ ./mygrep.sh -v testfile.txt
Error: Missing search pattern or filename.
Perhaps you meant to search for 'testfile.txt' in a file?
Run './mygrep.sh --help' for usage information.
(Note: The script should also exit with status 1 here)

$ ./mygrep.sh --help
Usage: ./mygrep.sh [OPTIONS] PATTERN FILE
Search for PATTERN in FILE case-insensitively.

Options:
  -n   Prefix each line of output with the 1-based line number
           within its input file.
  -v   Invert the sense of matching, to select non-matching lines.
  --help   Display this help message and exit.

Arguments:
  PATTERN The string to search for (case-insensitive).
  FILE   The input text file to search within.

Examples:
  ./mygrep.sh hello input.txt   # Find lines containing 'hello'
  ./mygrep.sh -n hello input.txt   # Find lines containing 'hello' with line numbers
  ./mygrep.sh -nv hello input.txt # Find lines NOT containing 'hello' with line numbers
(Note: The script should exit with status 0 here)

(Remember to take an actual screenshot of your terminal showing these commands and their output.)

🧠 Reflective Section

1. **Argument and Option Handling:**
    - **--help:** The script first checks if the *very first* argument is exactly --help. If so, it calls the usage function (which prints help and exits) and stops further processing.
    - **getopts:** The while getopts "nv" opt; do ... done loop processes short options (-n, -v). getopts reads options from the script's arguments one by one. If it finds -n or -v, it sets the opt variable to n or v respectively. The case statement then sets the corresponding flags (show_line_numbers=1 or invert_match=1). getopts allows options to be combined (like -nv). If an invalid option is found, getopts sets opt to ? and the invalid option character is stored in $OPTARG. The \?) case handles this, printing an error and exiting.
    - **shift $((OPTIND-1)):** After getopts finishes, the shell variable OPTIND holds the index of the *next* argument *after* the last processed option. shift removes arguments from the beginning of the positional parameter list ($@). shift $((OPTIND-1)) effectively removes all the options and their potential arguments (though we have none here) that getopts processed, leaving only the non-option arguments (expected to be PATTERN and FILE).
    - **Argument Validation:** The script then checks if exactly two arguments remain (if ["$#" -ne 2]). If not, it prints an appropriate error message (differentiating between 0, 1, or >2 remaining arguments) and exits with status 1. If two arguments remain, they are assigned to the pattern and filename variables.
    - **File Checks:** Finally, it verifies that the filename corresponds to an existing file

(-f) and that the script has read permissions for it (-r), exiting with an error if either check fails.

2. Supporting Regex or -i/-c/-l Options:
   Adding support for full regular expressions and options like -i (explicit case-insensitivity), -c (count matching lines), or -l (list filenames containing matches) would significantly change the script's core logic:
   - **Regex:** The simple string comparison [["${line,,}" == *"${pattern,,}"*]] would be insufficient. You'd likely replace it with Bash's regex matching operator ([["$line" =~ $pattern]]). Handling case-insensitivity with regex in pure Bash can be tricky (might need shopt -s nocasematch or more complex patterns), so often the simplest approach becomes calling the *actual* grep command internally (e.g., echo "$line" | grep -qE "$pattern") and checking its exit status. You might need another option (e.g., -E) to enable regex mode.
   - **-i (Explicit):** You'd add i to the getopts string. The matching logic would then conditionally apply case-insensitivity (e.g., using ${var,,} or passing -i to an internal grep call) only if the -i flag was set.
   - **-c (Count):** Instead of printing lines inside the while loop, you would increment a counter variable whenever a match (or non-match for -v) occurs. After the loop finishes reading the file, you would print the final value of the counter.
   - **-l (List Filenames):** Upon finding the *first* matching line (respecting -v), the script would print the filename and then immediately exit the while loop (using break) or even exit the script entirely (using exit 0) since no further processing of that file is needed. The match_found logic would still apply to set the final exit status correctly if no match was ever found.
3. Hardest Part to Implement:
   For this specific script, the most challenging part is often correctly integrating the getopts parsing with the subsequent argument handling (shift and validation) and ensuring the logic for combining the -n and -v flags works as expected in all cases. Specifically:
   - Understanding how getopts modifies OPTIND and using shift correctly is crucial but can be confusing initially.
   - The conditional logic to decide *whether* to print a line based on both the match result (if [[...]]) *and* the $invert_match flag requires careful nesting or boolean logic.
   - Applying the line number prefix ($line_num:) *only* when -n is active *and* the line is actually being printed (respecting -v) adds another layer of conditional checking.
   - Finally, mimicking grep's exit status (0 if lines were selected, 1 otherwise) requires tracking whether *any* output occurred (match_found flag) and setting the exit code accordingly *after* processing the entire file.