Word count exercise

Problem statement

Write a command line program that implements Unix `wc` like functionality. Read the man-page for `wc` in case you don't know what it does. Implement all options such as `-l` (show only line count), `-w` (show only word count), `-c` (show only character count). Also allow to combine these options as the user wants. If no option is specified, then display all three values (line, word and char count). The input to `wc` can be single or multiple files or even STDIN.

The program should have following features.

Story 1 (print line count for a single file)

Given a single text file as input, implement `wc -l` that counts the number of lines in the file. The program will be executed as follows:

```
# happy path example
$ ./wc -l file.txt
      19 file.txt

# error scenario 1
# note that the file protected_file.txt has no read permission for the user
$ ./wc -l protected_file.txt
./wc: protected_file.txt: open: Permission denied

# error scenario 2
# foo.txt file doesn't exist
$ ./wc -l foo.txt
./wc: foo.txt: open: No such file or directory

# error scenario 3
# bar is a directory, instead of a file
```

```
$ ./wc -l bar
./wc: bar: read: Is a directory
```

Assumptions:

- Your program behaviour should be exactly same as `wc` command behaviour.
- The output is printed on STDOUT.
- In case of error scenarios, program should fail with a non-zero exit code.
- The line count printed on the STDOUT should be right aligned and printed with 8 character width (same as that of `wc -l` command).
- The program is packaged as a standalone binary (`./wc`). Based on the programming language you're using, you may be able to create a standalone utility (e.g. in Golang). If not, use any CLI execution method in your language (e.g. java -jar wc.jar for Java)

Expectations:

- Write test cases for success and error scenarios.
- The program should print correct output (the output should be same as that of the actual `wc -l` command).

Story 2 (print word count for a single file)

Given a single text file as input, implement `wc -w` that counts the number of words in the file. Words are whitespace separated.
If you're in doubt what a "word" is in this context, refer to the actual `wc -w` output. Your program's output should match with actual `wc -w` 's output.

The program will be executed as follows:

```
# happy path example
$ ./wc -w file.txt
     102 file.txt
```

Assumptions:

- The word count printed on the STDOUT should be right aligned and printed with 8 character width (same as that of `wc -w` command)

Expectations:

- Write test cases for your code. Identify the edge cases (only whitespaces, emojis in the text, etc.)
- Try to reuse as much code as possible from the previous story.
- The program should print correct output (the output should be same as that of the actual `wc -l` command).

Story 3 (print character count for a single file)

Given a single text file as input, implement `wc -c` that counts the number of characters in the file.
If you're in doubt what a "character" is in this context, refer to the actual `wc -c` output. Your program's output should match with actual `wc -c` 's output.

The program will be executed as follows:

```
# happy path example
$ ./wc -c file.txt
    3581 file.txt
```

Assumptions:

- The character count printed on the STDOUT should be right aligned and printed with 8 character width (same as that of `wc -c` command)

Expectations:

- Write test cases for your code. Identify the edge cases (only whitespaces, emojis in the text, newlines, etc.)
- Try to reuse as much code as possible from the previous stories.
- The program should print correct output (the output should be same as that of the actual `wc -l` command).

Story 4 (print all line, word and character counts for a single file together)

Combine the previous three stories and print all the counts together. This is equivalent to running a `wc filename.txt` command.

The program will be executed as follows:

```
# happy path example
$ ./wc file.txt
      19      102     3581 file.txt
```

Assumptions:

- The counts printed on the STDOUT should be right aligned and printed with 8 character width (same as that of `wc` command)

- Line count should be printed first, followed by the word count and then finally character count (e.g. 19, 102, 3581 in above example)
- Each of the counts should be separated by a space. The actual `wc` command uses 8 char width for displaying the counts and just adds a space for filename.
  We want to add extra space to separate each of the counts. This is where the program differs slightly from the actual `wc` output.

Expectations:

- Write test cases for your code.
- Try to reuse as much code as possible from the previous stories.
- The program should print correct output (the output should be same as that of the actual `wc` command).

Story 5 (allow any combination of flags: -l, -w, -c)

As a user, I should be able to combine any or all the flags, in any order.

The program will be executed as follows:

```
# print only line and character count
$ ./wc -l -c file.txt
      19     3581 file.txt
# print only character and word count (order of the flags shouldn't matter). output
$ ./wc -w -c file.txt
     102     3581 file.txt
# print all counts, but with flags (this is same as running `./wc file.txt` commanc
```

```
$ ./wc -w -c -l file.txt
      19      102     3581 file.txt
```

Story 6 (Make it work on multiple files)

As a user, I should be able to invoke `wc` on multiple files at once. When executed on multiple files, the last line should print the total count from all files (same as the actual `wc` command)

The program will be executed as follows:

```
# happy path example
# run word count on multiple files
$ ./wc file1.txt file2.txt
      19      102     3581 file1.txt
       6       32      274 file2.txt
      25      134     3855 total

# error scenario 1
# pass a directory in between a few files
$ ./wc file1.txt dir file2.txt
      19      102     3581 file1.txt
./wc: dir: read: Is a directory
       6       32      274 file2.txt
      25      134     3855 total
```

Assumptions:

- A user should be able to combine the functionality (and error cases) from all previous stories if they want to. i.e.

    - Run `wc` on multiple files with specific flags (-c, -w, etc).
    - Run `wc` on some directories and/or files without read permissions.
    - Run `wc` on some non-existing files.
- In case of any error, the exit code from the command should be non-zero. Also, the error output should be printed on STDERR, instead of STDOUT.
- The total should be displayed on the last line.

Expectations:

- Write test cases for your code.
- Try to reuse as much code as possible from the previous stories.
- The program should print correct output (the output should be same as that of the actual `wc` command).

Story 7 (Allow input from stdin)

As a user, I should be able to invoke `wc` by passing input via stdin. When the program is executed without any filenames, it should read the input from standard input stream (stdin) and calculate word count based on that.
To indicate that the input has ended, user can press `Ctrl+d` .

The program will be executed as follows:

```
# run word count on stdin
$ ./wc
abc
def ghi jkl
```

```
^D (this is just to indicate that user has pressed Ctrl+d)
       2       4       16
```

Expectations:

- Write test cases for your code.
- Try to reuse as much code as possible from the previous stories. We want to see how you design your code to handle input from both files and stdin.
- The program should print correct output (the output should be same as that of the actual `wc` command).

Story 8 (Performance and Optimisations)

Think about the performance and optimisations that you can do in your code to handle "scale". Some points you can think about:

- Does your program work as expected when we run it in a directory containing thousands of files? Does it process one file at a time?
  Can you think of any performance optimisations where you spin up one goroutine per file?
- Does it handle large files e.g. files with 10+GB size. Are you reading the entire file in memory and then calculating the lines, words and chars?
- Can you read files line by line instead of reading it entirely in memory? What changes will you have to do in your program to support this?
- Load test your program on these files https://github.com/ravexina/shakespeare-plays-dataset-scraper/tree/master/shakespeare-db

Even if you don't implement this story, write down your thought process, assumptions, code design in a few bullet points and add that to the readme.

Feel free to make suitable assumptions if needed, ensure to document them in README.md