IC221: Systems Programming
Project 1: The Word Count Program
Spring, AY22

Project Learning Goals

- Write C programs effectively
- Work with command line arguments
- Read from files and stdin to complete tasks
- Write functions that handle structured data

Grading Rubric (100 points total)

(10) Complete `man` page for your program. Proper spelling and grammar for full credit.
(10) Proper error reporting for all command-line and file-not-found errors.
(20) Correct command-line parsing and processing of `stdin`, a single file, or multiple files
(20) Line count using `fgetc()` (half credit for using `fscanf()`)
(20) Word count using `fgetc()` (half credit for using `fscanf()`)
(20) Character count using `fgetc()` (half credit for using `fscanf()`)
For full credit with `fgetc()`, make only a single pass reading through each file.

Submission

- Source code: `wc.c`
- Manual Page entry for your program: `man.txt`

Description

In this project, you will write aprogram that functions similarly to the command line `wc` utility, short for "word count." The `wc` utility can also count the number of characters and lines in a file.

Example output:

```
$ ./wc dickens.txt
dickens.txt 19199 161007 917028

$ ./wc A.txt
A.txt 1 3960 201961

$ ./wc random.txt
random.txt 69765 295983 6795742
```

The first number is the number of lines, the second number is the number words, and the third number is the number of characters. If multiple files are specified, the totals in each category are also reported, after the individual file results (see highlight):

```
$ ./wc dickens.txt A.txt
dickens.txt 19199 161007 917028
A.txt 1 3960 201961
total 19200 164967 1118989
```

Input

```
wc [-c -l -w] [file ...]
```

If no files are specified, then `wc` should read from `stdin`:

```
$ cat dickens.txt | ./wc
-stdin- 19199 161007 917028
```

You should indicate "`-stdin-`" for the filename in this case.

A user can also indicate that they wish to read from `stdin` using the + symbol as a file name, as in the following:

```
$ cat dickens.txt | ./wc A.txt + A.txt
A.txt 1 3960 201961
-stdin- 19199 161007 917028
A.txt 1 3960 201961
total 19201 168927 1320950
```

Options

Prior to the list of files, optional command line arguments can be provided to limit the output to just reporting the number of lines, number of words, or number of characters (or some combination):

```
-l print number of lines
-w print number of words
-c print number of characters
```

Examples:

```
$ cat dickens.txt | ./wc -l A.txt + A.txt
A.txt 1
-stdin- 19199
A.txt 1
total 19201

$ cat dickens.txt | ./wc -w A.txt + A.txt
A.txt 3960
-stdin- 161007
A.txt 3960
total 168927

$ cat dickens.txt | ./wc -c A.txt + A.txt
A.txt 201961
-stdin- 917028
A.txt 201961
total 1320950
```

Command line arguments can be combined as well:
```
$ cat dickens.txt | ./wc -c -l A.txt + A.txt
A.txt 1 201961
-stdin- 19199 917028
A.txt 1 201961
total 19201 1320950
```

Output is always reported the same order -- lines, words, characters -- regardless the order of the command line arguments.

All command-line options must precede *all* filenames.

Command line options must be separated: instead of `-wlc`, use `-w -l -c`.

How to Count

Characters: Include all printable and non-printable characters. For example, increment every time `fgetc()` retrieves a new character. Your output should match the output of `wc -c` using the built-in command line program. Your program will only be tested using text files that contain only printable ASCII characters (including spaces and tabs) and LF ('`\n`') characters.

Lines: Increment the line count each time a '`\n`' is encountered. Your output should match what is reported using the built-in `wc` command, as in `wc -l`.

Words: There are two methods for how you can choose to count words. However, for full credit you must use `fgetc()`. Let's start with a simpler method, using `fscanf()`.

You can use `fscanf()` to read a file multiple times, with different format characters on each pass, to determine line, word, and character counts. For words, your can use the "`%s`" format which will recognize word boundaries, but you will need to specify a buffer large enough to store the resulting word, which may fail for large words. You could then read lines and chars by using the "`%c`" format to count characters and detect only newline ('`\n`') symbols.

A more efficient method is to use `fgetc()` which reads from the specified file one character a time. The challenge with this method is then you need to a way to detect word boundaries. To do that, you should employ the `ctype.h` library and the `isspace()` function. Using this method, you should be able to make a single pass through a file and perform all your counts.

How to Count Words with `isspace()`
Counting characters and lines are both pretty straightforward. When counting *words*, your program should function similar to the built-in program `wc`. For example:

- A word *begins* when the current character being scanned is printable (not whitespace), and the prior character is a whitespace character (see reference to `isspace()`, below).

- A word *ends* when the current character being scanned is a whitespace character, and the prior character is printable (not whitespace).

- The word count should not change when the current and prior characters are both whitespace, or when the current and prior characters are both non-whitespace.

Parsing Command Line Options

A full solution to this project must be able to handle command lines. The parsing requirements for command line options are as follows:

- Command line options must come *before* the list of files. If a command line option appears within the list of files, it is treated like another filename.
- Command line options must begin with a - (tack/hyphen).
- You should report an error on unknown options (see below).
- Once you reach the first command line argument *without* a -, you can assume the list of files has started and process everything remaining as a filename.

Error Conditions

Report errors on incorrect user input. There are two main error conditions:

(1) Unknown *filename*: You should report the error *only* to `stderr`, but continue to proceed with processing remaining files. Do not exit. Use the following format, including the real filename that could not be opened:
```
ERROR: file <filename> cannot be opened
```

(2) Unknown *command line argument*: You should report the error to `stderr` then exit with an exit status code of 2. Use the following format, including the actual invalid option that the user typed:
```
ERROR: unknown option <option typed by user>
```

All error reporting should be done to `stderr.` If the program completes normally, use an exit status code of 0.

Some examples of condition (1) errors:

```
$ ./wc doesnotexist.txt
ERROR: file 'doesnotexist.txt' cannot be opened

$ ./wc doesnotexist.txt 2> /dev/null

$ ./wc dickens.txt doesnotexist.txt dickens.txt
dickens.txt 19199 161007 917028
ERROR: file 'doesnotexist.txt' cannot be opened
dickens.txt 19199 161007 917028
total 38398 322014 1834056

$ ./wc dickens.txt doesnotexist.txt dickens.txt 2>/dev/null
dickens.txt 19199 161007 917028
dickens.txt 19199 161007 917028
total 38398 322014 1834056
```

Some examples of condition (2) errors:

```
$ ./wc -p dickens.txt
ERROR: unknown option '-p'

$ ./wc -p -l dickens.txt
ERROR: unkown option '-p'

$ ./wc -l -p dickens.txt
ERROR: unknown option '-p'
```