CSE 303, Winter 2006, Assignment 5A
Due: Wednesday 22 February, 9:00AM

Last update: 12 February

You will write some I/O code and unit-tests for it while other group members *independently* implement a word-count data structure and some "counter distance" code. The sample solution is 60–65 lines, *not including* testing code.

Requirements:

- Put your code in two files, `5a.c` and `5a_test.c`. Both should include `5a.h`, which you should write. `5a.h` needs just this prototype and typical header-file stuff:

  ```
  int next_multiletter_word(FILE * file, int buflen, char * buf);
  ```

- In `5a.c`, implement `next_multiletter_word`. Use helper functions as appropriate, of course.
- Assume `file` is a file opened for reading, and `buf` points to an array large enough to hold `buflen` characters.
- Your function should read the next "word" from the file as follows:
  - Skip any characters that are not an English letter (either case) or the numbers 0 or 1. That is, the word does not start until such a letter is reached.
  - The word continues until a character that is not an English letter (either case), 0, 1, a period, or a comma is reached.
  - In the word, any period or comma is skipped. For example, the character sequence `a..r.t.f,ul.` is the word `artful`.
  - Skip one-letter words. For example, `a`, `e`, and `f.` should all be skipped – it is like the one English letter is a space. But `0a`, `01`, and `a.0` are all two-letter words.
- "Convert" the word you find as follows:
  - Remove every period and comma.
  - Convert every capital letter to lower case.
  - Convert every `0` to `'o'`.
  - Convert every `1` to `'l'`.
- If you reach an `EOF` character (end-of-file) without finding a word, return -1.
- If the next (converted) word *plus* a trailing `'\0'` would fit in `buf`, then store it and a trailing `'\0'` in `buf` and return the length of the word (not including the trailing `'\0'`).
- If the next (converted) word *plus* a trailing `'\0'` would *not* fit in `buf`, then still return the length of the word (not including the trailing `'\0'`). (So the return value does not depend on `buflen`.) But also *reset* the "current position" of `file` to where it was before the function was called. If resetting the position fails, print an appropriate message to `stderr` and exit the program.

  You may set contents of `buf` to whatever you want in this case, but you must not write too many characters.
- In `5a_test.c` put unit tests for your code and a `main` that runs them. You will want to provide also text files for reading, which your test code will open.

Advice:

- Use `man` or a reference manual to learn about `fgetc`, `fseek`, `ftell`, `isalpha`, and `tolower`.
- First store the starting position of the file in case you need to reset it later.
- Then find the first and second letter of the word (since you skip one-letter words).
- Checking that `buflen` is large enough, store these letters (after conversion) in `buf`.
- Then loop until the end of the word, checking `buflen` as appropriate and converting each character as you go (and skipping periods and commas without storing them).
- Remember that if you encounter `EOF` before a two-letter word, you return `-1`, but otherwise `EOF` may mark the end of a word that should still be returned.
- Because much of the character-checking for the "first two characters" and the "remaining characters" is the same, helper functions will prove useful.

Assessment and turn-in:

Your solutions should be:

- Correct C code that compiles without warnings using `gcc -Wall` and does not have space leaks
- In good style, including indentation and line breaks
- Of reasonable size

Your test code should provide good *coverage*.

Turn in your example text files.

Use `turnin` for course cse303 and project hw5. If you use late-days, use project hw5late1 (for 1 late day) or hw5late2 (for 2) instead of hw4.