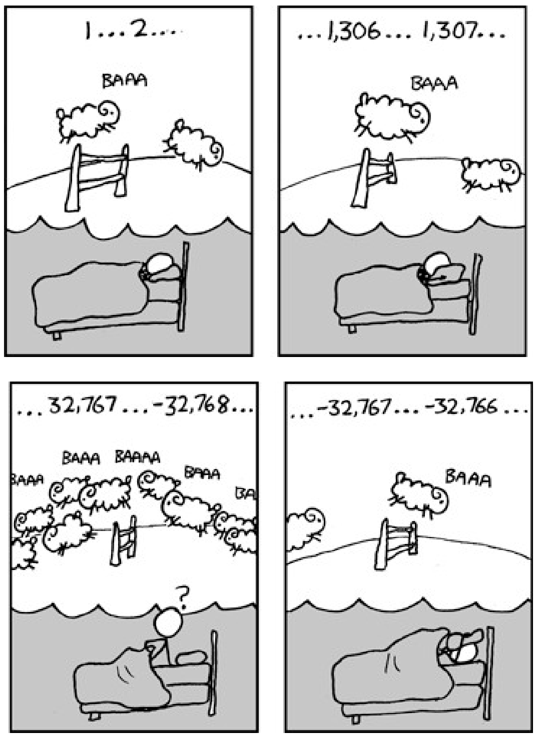
CS 218
Homework, Asst. #11

Purpose: Become more familiar with operating system interaction, file input/output operations, and file I/O buffering.
Due: Tuesday (3/26)
Points: 250 (grading will include functionality, documentation, and coding style)

<u>Assignment:</u>
Write an assembly language program that will read a file and count the the number of occurrnaces of a user-specified word. The count will be displayed in ASCII/Dozenal format. The program should read the word, case flag, and file name from the command line. The program must perform error checking. If there is an error, appropriate error message should be displayed and the program terminated. An example command line arguments is:

```
./wordCount -w the -ic -f file.txt
```

To ensure efficiency, the program must perform buffered input. The buffer size should be set to BUFFSIZE which is declared as a constant (i.e., BUFFSIZE equ 300,000). *Note*, the buffer size will be changed in the next assignment, so all code should reference the constant.



Source: www.xkcd.com/571

The provided main program calls the following routines:

- A boolean function, **checkParameters()**, that reads and checks the command line arguments. The command line arguments must be in the format of: **`-w <searchWord> <-ic|-mc> -f <inputFile>`** in that order. The search word must be of non-zero length. To check the file name, attempt to open the file. If the file opens, the routine should return the file descriptor. If there is an error, an appropriate error message should be displayed (see examples). The function should return TRUE or FALSE.
- A boolean function, **getWord()**, that should return a single word from the text file (thus, skipping white space as **`word >> inFile`** in C++ would do). White space includes tabs, spaces, LF, etc. (any character ≤ to a space). In order to perform this efficiently, the routine must perform input buffering. As necessary, data from the file should be read into a primary buffer (BUFFSIZE large). A single word of text should be returned to the calling routine (from the primary buffer) which must be < the passed maximum word length. When the primary buffer is empty, the routine should fill the primary buffer by reading the file. The function must handle any unexpected read errors. The function should return TRUE or FALSE.
- A void function, **checkWord()**, that compares the search word to the current word and increments the count if they match. If the match case flag is TRUE, the search word and current word must match exactly. If the match case flag is FALSE, the words should be considered matching regardless of case. For example, 'Section' matches 'section' when ignoring case.

- A **displayResults()** function to write the final results to the screen. Spefically, the message should be "Found '<searchWord>' <dozenalNumber> times". Refer to the sample output for formatting examples.

A main program and a functions template will be provided. All functions must be in a separate source file that is independently assembled. Only the functions file, not the provided main, will be submitted on-line. As such, you must not change the provided main!

Submission:

When complete, submit:

- A copy of the **source file** via the class web page (assignment submission link) by 11:55 PM. ***Assignments received after the due date/time will not be accepted.***

Debugging -> Command Line Arguments

When debugging a program that uses command line arguments, the arguments must be entered after the debugger has been started. The debugger is started normally (ddd <program>) and once the debugger comes up, the initial breakpoint can be set. Then, when you are ready to run the program, enter the command line arguments. This can be done either from the menu (Program -> Run) or on the GDB Console Window (at bottom) by typing `run <commandLineArguments>` at the (gdb) prompt.

Testing

Note, a utility for testing will be made available on the class web page. A script file to execute the program on a series of pre-defined inputs will be provided. *Note*, please follow the I/O examples. The test script compares the program output to pre-defined expected output (based on the example I/O).

Example Executions:

The following will read file "a11f3.txt" file and count the occurrances of the word 'section' while ignoring the case and then matching case.

```
ed-vm% ./wordCount -w section -ic -f a11f3.txt
Found 'section' 1X times.
ed-vm% ./wordCount -w section -mc -f a11f3.txt
Found 'section' 0 times.
```

The following are some additional examples, including error handling:

```
ed-vm% ./wordCount -w section -ic -f nofile
Error, can not open input file.
ed-vm%
ed-vm% ./wordCount -w -ic -f a11f3.txt
Error, invalid command line arguments.
ed-vm%
ed-vm% ./wordCount -w section -i -f a11f3.txt
Error, invalid match case specifier.
ed-vm%
ed-vm% ./wordCount -wrd section -mc -f a11f3.txt
Error, invalid search word specifier.
ed-vm%
ed-vm% .wordCount -w and -ic -f a11f3.txt
Found 'and' 19X times.
ed-vm%
```