# File Handling Activity

Program 1: [ 10 points ] Write a program to echo the "argument string" entered on the console to the screen and possibly to a file. The file name can be passed into the program from the command line using the -f option. The argument string entered will be written in all upper or lower case based on options, or mixed case as the default.

Usage: writer  [-lu] [-f filename] "argument string"

* The -l option is to print in lower case.
* The -u option is to print in upper case.
* If both the -l and -u option are included then there is an error. Tell the user and print the usage then quit.
* If the -f option is included with a filename, then create a file with that name or overwrite a previously existing file and write your name in on the first line followed by the "argument string". Also write "argument string" to screen.
* If the -f option is not included, then only write your name to the screen followed by "argument string".   Therefore "argument string" is mandatory to be displayed on screen.

Use a copy of the original getopt to parse the command line.

the "argument string" can be a sentence of 3-10 words, 80 characters max, mix upper and lower case. Check the limits.

Be sure to test whether a file open succeeds or not and close any file you open when done.

Use stdlib file IO -  fopen, fprintf, fflush, fclose etc. [Not System Calls].

Examples:

.writer -l -f foo.txt is a beautiful day

.writer -u -f foo.txt is a sad day

.writer -f foo.txt is a happy day

.writer -lu -f foo.txt this is a bad command

.writer -l you should see this on the screen

.writer -x -f foo.txt is this a good command

Program 2:  [10 points) Repeat a similar program called the "updater" but this time append to the newly created file you opened the passed "argument string".  Also, run program multiple times using different "argument strings" to fill the file with at least 5  "argument strings" and no more than 10.   In this case the argument string should be exactly a 5 word sentence.  No need for -l or -u in this "updater" program.  No need to write your name to the file.

Usage: updater   -f filename "argument string"

* -f is mandatory
* no need to write to screen

Be sure to submit the file you write on program 2 with the zip.

Examples:

.updater -f bar.txt this is the first sentence

.updater -f bar.txt this is the 2nd sentence

.updater -f bar.txt this is the 3rd sentence

.updater -f bar.txt this is the 4th sentence

.updater -f bar.txt this is the 5th sentence

.updater -f bar.txt this is a very long sentence that should not work

.updater why is this not providing a filename

Program 3:[10 points]  Write a program that opens the file created in Program 2 call it "reader".  Pass a parameter to indicate lines to read. Read all the lines in the file based on the parameter 'lines'  into a buffer and using sscanf parse all the words of the sentences into different string\* variables.   Print each of the words to the screen followed by a carriage return.  You can hardwire filename or use option to enter filename from Program 2.

Usage: reader  [-f "filename"]  'lines'

* -f is optional (if not provided have a hardwired name from previous program 2 (bar.txt))
* 'lines' integer from 1 to 10 (remember the files has 5-10 strings from previous program)

For example:

Opening file "bar.txt"

File Sentence 1 word 1 = Hello

File Sentence 1 word 2 = Students

....

File Sentence 5 word 5 = Class

Examples of commands:

.reader -f bar.txt 5

.reader -f bar.txt 2

.reader -f bar.txt 7

.reader 2

.reader 5

.reader 8

.reader 11