SYSTEM PROGRAMMING MIDTERM
Spring 2010

1.

a Write a script that takes three command line arguments. The first argument is a file name. The second and third arguments are words (ie, strings of alphabetic characters.) Your script will complain if there are not exactly three arguments – and then it will exit with an error value. Your script will next test to see if the file exists and to see whether the two words are, in fact, just strings of alphabetic characters. If any of these things are not true, your script will prompt the user to reenter the incorrect argument. Then it will retest it.

Here is an example of how the script would work:

```
% ./myscript F1 word w7rd
Error. File "F1" does not exist. Please enter a new name.
> File1
Error. "w7rd" is not a word. Please enter a new word.
> word2
Error. "word2" is not a word. Please enter a new word.
> otherWord
%
```

b Assume that you have the script from part a. So you know that the file exists and that the two words are actual alphabetic strings. Now you want to look for all lines file which contain the first word followed (not necessarily immediately) by the second word. You will use *sed* to reverse these words. Note that these words must be whole words in the line, in order to match. Not also that you only do the first match per line.

Here is an example of how the script would work:

```
% cat File1
bird cat lion. cat lion.
lion cat
cat cat lion
A cat and a bunch of lions.
Cat is like a lion.
the cat said, "lion."
"Scat," said the lion.
% ./newscript File1 cat lion
% cat File1
bird lion cat. cat lion.
lion cat
lion cat cat
A cat and a bunch of lions.
Cat is like a lion.
the lion said, "cat."
"Scat," said the lion.
%
```

c Now we want to allow the arguments to be partial words, but we want to swap the whole word that they belong to. Show the new sed command.
Here is an example of how the script would now work:

```
% cat File2
Boating upriver is fun
boating upriver is fun
A "boat" can "float" on a "river."
I like to take my boat out on the river
out rivering on my boat.
% ./partialwordscript File2 boat river
% cat File2
Boating upriver is fun
upriver boating is fun
A "river" can "float" on a "boat."
I like to take my river out on the boat
out boat on my rivering.
%
```

d Explain why sed's "/g" option would not be good enough to swap multiple times. (That is to say, the first instance of word1 swaps with the first instance of word2, the second instance of word1 swaps with the second instance of word2, etc.)

e Suppose we want to swap the two words, regardless of which one comes first. Explain why you can't just say:

```
./newscript File1 cat lion ; ./newscript File1 lion cat
```

g Going back to part a, we have a file and two words. Write a *sed* command to display all lines between: 1) the first line that starts with word1 and 2) the first subsequent line that ends with word2.

2.

a Write a script that takes one command line argument. It then prints the letters of the argument, one letter per line:

```
% ./q2script all-in-1
a
l
l
-
i
n
-
1
%
```

b. Extend this script so that it prints the word back in reverse:

```
% ./q2script all-in-1
1-ni-lla
%
```

3. Write short commands to do each of following to the input stream:

a. Capitalize all letters

b. Replace all non-alphanumerical characters with a newline

c. Print all nonempty lines

d. Count the number of empty lines

e. Save to a file named "SaveFile" and pass to the output stream

f. Print lines that contain a decimal number, such as 5.687 or -0.3, as a whole word. (Note the decimal point is required. Note also there must be at least one digit before and after the decimal point.)

g. Execute a command, "./com1". If it is successful, execute a second command, "./com2".

h. Grep for the phrase that begins with z at the beginning of a line and ends in -ic or -ics at the end of the line, and which has gm in it somewhere.

i. Show the lines that contain the phrase "^\\.^$" – but the phrase cannot be at the beginning or the end of the line. Please note: even if the phrase is inside the line, it is still invalid if it is also at the beginning or end. Example: "This ^\\.^$ is bad^\\.^$"

j. Show the line numbers (just the numbers) of all lines containing the word "thou." It should not find the word "though." It should find the word "Thou."

k. Lines containing only a single lower case letter. Example: "THIS ^ Is GOOD.. %"

l. An egrep that will find lines containing any of the following: 95060, 95062, 95064, 95065, 95066,95005,95017,or 95018.

m. Print all of the unique lines. Order does not matter.

n. Print: He said, "I overheard you when you said, 'No!'"

o. Display the contents of all of the files whose names are passed in on the input stream

p. Find an "a" followed later (but not immediately) by a "b".
So these lines match: "abad ball" and "a7b" match
But these lines do NOT: "baby", "abe b", "apple", "bay"

q. Find lines that contain one of the following subsequences: "BAKE", "BIKE", "BATE", or "BITE". But these are subsequences not substrings. So all of the following are matches:
This line B mATchEs
So does this BI line TTE.
And so does this one BIKE
BAnd this KE one

4. Write the ls command to list all files that have an "a" followed later by a "b". You are allowed to have a "b" immediately after the "a". But that "b" will not count in the match.

So these files match: "abad ball", "a7b", "abe-b"
But these files do NOT: "baby", "apple", "bay"

5. Write a script to: 1) make a list all of the integers in a file, 2) add up the sum of these integers. You must use foreach and ` in your script.
Note that an integer cannot be followed by a decimal point, but it can be followed by a period if it is the end of a sentence. These integers should be whole words in the file. They may have a negative sign.
In other words: "5, -3, and 8." ⇒ "5 -3 8". But "5.0, -3x, and x8." ⇒ "".

6. Describe what each of the following lines (from tcsh scripts) does, in 1 or 2 sentences.

a. `if ($word == $argv[$#argv]) then`

b. `ls -l ./*/*/*.c`

c. `sed '2,$ s/.*/&/p'`

d. `sed -e 's/\([0-9]*\):.*/\1/'`

e. `sed 's/^[A-Z]/| &/'`

f. `sed -e 's/\n[0-9]*\:/|/'`