SYSTEM PROGRAMMING MIDTERM
Spring 2011

Note: This entire exam requires that you are in the tcsh shell.

1. (40pts) There is a file in the current directory, named file, with the following contents:

```
% cat file
cat dog mouse
                                   ← This line is empty
Cat Dog Mouse
cat, cat.
cats cat $cats.
```

What is the output of each of the following commands?
Note: *Parts a-t are independent, so, for each one, the contents of* file *are as shown above.*
Note: *If you have any empty lines in your output, indicate them with a "_" on that line.*

a. `cat file | sed 's/cat/dog/p'`

b. `grep -vw cat file`

c. `cat file | tr " " "\n" | grep cat`

d. `cat file | tr -dc "a-z"`

e. `head -1 file | sed 's/dog/cat/g; s/cat/dog/g'`

f. `cat file | sed -n 's/dog/DOG/p; s/cat/CAT/g'`

g. `cat file | sed -n '/dog/ p; =; i *'`

h. `echo "cat file" | xargs grep`

i. `head -1 file | tee file | cat; tail -1 file`

j. `head -1 file | tee file | cat || tail -1 file`

k. ``head -n `cat file | wc -l` file | wc -l``

l. `cat file | cut -c1-3 | uniq`

m. `cat file | sed -n 'n; p'`

n. `cat file | sed -n 'x; n; p; g; p'`

o. `grep -n "[dD]og" file | sed -e 's_\([0-9]*\):.*_\1_' | tr "\n" " "`

p. `cat -n file | sort -gr`

q. `grep "." file`

r. `grep "[.]" file`

s. `echo file | wc`

t. `set x = "cat"; grep '$cat' file`

2. (26 pts) Assuming that you are receiving input from a pipe (eg: cat file | <mycomand>), write a **single instruction** to perform each of the following tasks:

a. To display the sentence: She said, "He said, 'No way!'"

b. To convert the input into ROT-13 encoding. (In ROT-13 encoding, each letter is rotated 13 positions. So a→n, b→o, …, m→z, n→a, o→b, …z→m, A→N, …)

c. To list all of the files in the current directory that either: 1) contain the substring **ab**, or 2) begin with a **number**, or 3) contain an **f** followed 2 characters later by a **g**, or 4) contain internally a **h** (that is, not counting the beginning or end). Note: *If a pattern matches multiple rules, it is OK if it lists multiple times.*
Note: *If you use 4 commands for the 4 possibilities, you will get most of the points.*
For example, if the directory has some files with the following names, then the output of your command could be (highlights added by me for clarity):
abc **2**b **f**i**g** a**f**i**g**b a**h**b h**h**h aa**ab** a**ab**a **2**ab 2**ab**

d. To display all lines from the input that either: 1) contain the substring **ab**, or 2) begin with a **number**, or 3) contain an **f** followed 2 characters later by a **g**, or 4) contain internally an **h** (that is, not counting the beginning or end).

e. To display all lines that end with the word stored in X, followed by a period. If the word stored in X is “happy”, for example, then lines like these would display:
They were **happy.**
We are **happy.**

f. To find lines containing any of the following: 95060, 95062, 95064, 95065, 95066, 95005, 95017, or 95018. *Note: Your answer must be as short as possible.*

g. To print quotations within quotations on one line. For example, the input lines:

```
Joe said, "I heard that Mike said, 'That is OK.' So I guess it is OK. "
"But I heard Mike say, 'No,'" I replied.
"This line has" 'NO' output.
"This line has" 'YES' "output."
```

Will yield the output:

```
'That is OK.'
'No,'
'YES'
```

Note: *You may assume that the ' and " symbols are only used for quotations.*
(So you won’t have phrases like this: 'That's OK,' which has 3 ' symbols.)

So, formally, here is what you are looking for: A region inside of " symbols which itself contains a region inside of ' symbols. If you find this, then you print the inner region, along with the ' symbols.

h. In part g, the line:

```
"This line has" 'YES' "output."
```

Really should not have printed, because the YES is not inside of a *pair* of " symbols. So now extend the expression from part g, so that it excludes lines like this, by ensuring that the part inside of ' symbols comes after an odd number of " symbols.

3. (15 pts) Write a **single instruction or a single pipe of instructions** for each of the following:

a. 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:

```
^\\.^$  is bad^\\.^$ even though the phrase is in the middle.
```

b. 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.”

c. Count the number of empty lines.

d. Identify all hyphenated words in a file. For example, if you have the following file:

```
% cat file
There are many hyphenated words
in this file.
This ultra-fast-hardening glue is
a good choice for water-resistant-
bonding applications. Our no-risk, money-
back guarantee is out-of-this-world.
```

Then the output of your pipe is:

```
ultra-fast-hardening
water-resistant-bonding
no-risk
money-back
out-of-this-world
```

Note: *There are no punctuation symbols in the output, except for the hyphens.*
Note: *Hyphenated words can cross line boundaries.*

e. To put every character from the input file on its own line. For example:

```
% echo "a cat." | <your pipe here>
a

c
a
t
.
%
```

f. Regardless of whether you got part e to work, let’s assume that you did, and that saved it as a script called *parte*. Now use this script, in addition to other commands, to reverse a file. For example:

```
% head -2 file
There are many hyphenated words
in this file.
```

```
% head -2 file | <your pipe here>
.elif siht ni
sdrow detanehpyh ynam era erehT
```

Note: *You may <u>not</u> use cat –r or anything else that we did not teach it in this course.*
Hint: *Use sort –r, along with other things.*

4. (19 pts) Write tcsh scripts for each of the parts below. The names of the scripts are parta, partb, partc, etc. Because these parts are named in this way, you can use them in your later parts. This is good, because it means that you can make partc work, even if you failed to get partb to work (for example).

a. Write a script to shift over any input flags and then print the remaining arguments
Here, a flag is identified by its first character being a "-".
Here also, flags must come before the other arguments

b. Write script that first runs parta and captures the result. Using this result, it then checks to see if there are either two or three arguments remaining. If not, then an error message is printed and the script exits with a value that indicates an error.
If yes, then the script prints the arguments captured from the output of parta.

c. Write script that first runs partb and captures the result. If partb failed then exit with a value that indicates an error.
Otherwise, check to see whether the third argument, if present, corresponds to the name of a file currently in the current directory. If not then print an error and exit with a value that indicates an error.
Otherwise, create a file of some name which contains either the contents of the input stream (if two arguments) or of the file indicated by the third argument.
Print the first two arguments.

d. Write a script that first runs partc to find your two arguments and to prepare your file. If there is no problem, then this script should then list all lines between matches to argument 1 and matches to argument 2.

Different from parts a-d:

e. Write a script to make a list (one per line) all of the numbers in the input stream. The numbers may be integers and they may be floats. There are four allowed styles as shown in the examples here: 1, 123, -214, 1.43, -2.41. Note that there may be many other characters in the file, but these should be ignored.

We now want to calculate the product of the numbers found in parte. But we have a problem, because expr only works on integers. So:

f. Write a script that runs parte, takes the output and creates a new output without the decimal point. This new script also returns a number which indicates the total number of decimal place digits in the list. For instance:

If the input was:	-6.0200	4
12	*Then the output would be:*	-60200
10.3	12	*And the return value*
4.	103	*would be:*

5	(because the decimal point goes 5 from the right)

g. Write a script that runs partf and then calculates the product of all of the numbers returned from partf. You must use foreach and ` in your script. Before printing the final answer, you must add a decimal point the appropriate number of places from the right.