SYSTEM PROGRAMMING FINAL
Spring 2010

In this exam, you will get the majority of the points on a problem if you can come up with a solution that works and that follows the specified instructions for that problem.[1] But if you want all the points, your solution should not be unnecessarily long.[2] You might get a few bonus points if your solution is particularly elegant or readable.

1. Describe what each of the following lines (from tcsh scripts) does, in 1 or 2 sentences. You will get zero points if all you do is repeat the question.[3]

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. `set W2blanked = `echo -n $3 | tr -c "\v" "\v"``

g. `while(! -z differences)`

h. `sed "/$X/ { N; N; s/^\([^\n]*\)\($X\)\([^\v]*\)"'$'"/\1$2\3/ }"`

i. `cat newlies | head -`expr "$a" - 1` | tail -1`

j. `cut -c `expr $f1 '-' $fromtop`-$f1 f1`

k. `sed "s/\.\.\./^[[2;37;40m...^[[1;37;40m/g"` [4]

l. `sort -g | uniq`

2. You have a file that contains three kinds of lines: "D: ", "T: ", or "F: ". Any lines that do not begin with one of these letters should be ignored.

Lines that begin with D: are definition lines; they always have exactly two words after the D:. The first word is in English and the second word is in alphabetic Chinese. Both words are always lower case.

Lines that begin with F: will contain a series of English words (all lower case, without punctuation). The "F" stands for "From", because these lines need to be translated *from* English.

Lines that begin with T: will contain a series of Chinese words (all lower case, without punctuation other than the "-"). The "T" stands for "To", because these lines need to be translated *to* English.

I anticipate some questions:
Q: What about repeated definitions?
A: Every English or Chinese word that appears on a D: line will be unique. There will never be another definition containing that word.

---

[1] For example, if the problem calls for an AWK one liner, then I should only see: one line, one awk command, and no pipes. Or if the problem calls for a single SED command, then you can only have one sed command and no pipes (But you could use the ";" operation, because that does not require a new SED command.) Of course, some problems don't mention these kinds of restrictions.

[2] In most cases, I won't be requiring the absolutely minimum-length answer. But just that the length should be relatively close to it and not needlessly complex.

[3] For example, on number a, an answer of: "This tests to see if the value of the variable $word is the same value as the $#argvth elements of the $argv array". That answer will get you zero points. Instead you must demonstrate that you understand what it actually does.

[4] Here a ^[indicates the special escape sequence key

Q: What about incorrect lines?
A: There are no incorrect lines. You always know that a D: line contains three fields, and no lines have any capitalization or punctuation (except for the "-" in Chinese words).

Q: What about empty T: or F: lines?
A: These lines are never empty.

2a. You are to write a **single AWK program** that reads an input file of the above format and prints out the translated versions of the F: and T: lines. Words that are not in the dictionary will be left untranslated.

For example:

```
% cat infile
D: hello nee-how
D: his ta-de
D: you nee
D: we wa-mun
F: we said hello to his puppy
D: puppy shao-go
X: aaa bbb
T: nee shao-go x nee
F: we said hello to his puppy
%awk –f myprogram < infile
wa-mun said nee-how to ta-de puppy
you puppy x you
wa-mun said nee-how to ta-de shao-go
%
```

Notice, in this example, that the word "puppy" is not translated the first time, because it was not yet in the dictionary. But then it is translated in the later sentence, because it is now in the dictionary.

2b. We now want to extend the program from part a, to allow for whole sentences to be translated. To assist with this, write a **single AWK program** that given a line of words: 1)inserts a space before any punctuation symbol except the "-", 2) inserts a space after every upper case letter, followed by the lowercase version of that letter.

For example:

```
% cat infile2
F: we said hello to his puppy
F: We said, "hello" to his puppy
F: He shouted, "HURRY!"
F: After that, Adam and I will leave.
%awk –f mynewprogram < infile2
F f : we said hello to his puppy
F f : W we said ,  "hello " to his puppy
F f : H he shouted , "H hU uR rR rY y ! "
F f : A after that , A adam and I i will leave .
%
```

3. What is the output of the following code:

```
% cat code
#!/bin/usr/tcsh
# This$ is an odd$ program
# toddler
# odd
tail –5 code | head -1
grep “$1\$” code
tail -3 code
exit 0
A
B
C
D
E
% ./code odd
```

4. 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. Execute a command, “./com1”. If it is successful, execute a second command, “./com2”.

g. 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.

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

i. Print: He said, “I overheard you when you said, ‘No!’”

j. 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

k. An Awk One-liner to double space a file which already has blank lines in it.

l. An Awk One-liner to count lines. (emulates "wc -l")

m. An Awk One-liner to print the sums of the fields of every line.

n. An Awk One-liner to print the last field of each line.

o. An Awk One-liner to print every line with more than 4 fields.

p. An Awk One-liner to align all text flush right on a 79-column width.

q. An Awk One-liner to, if a line ends with a backslash, append the next line to it.

r. An Awk One-liner to switch the first 2 fields of every line.

s. An Awk One-liner to print the last 2 lines of a file. (emulates "tail -2")

t. An Awk One-liner to delete ALL blank lines from a file.

u. An Awk One-liner to remove duplicate, nonconsecutive lines.

v. An Awk One-liner to put sentences on lines (assuming that all sentences end with periods).

w. An Awk One-liner to reverse the order of every 2 lines. (An odd final line prints as is)

For example: echo "1 2 3 4 5"|tr " " "\n" | awk '...' | tr "\n" " " => 2 1 4 3 5

x. A sed command to insert line numbers (emulates grep –n "^").

y. A cut command to print columns 2 to 4

z. A lex program to compress blank spaces and tabs. (Multiple spaces/tabs compress to one space, and spaces at the end of the line are removed.)

5. Look at the following Makefile:

```
f1: f2 f3 f4
        ./com1
f2: f3 f5
        ./com2
f4: f6
        ./com3
f5: f7
        ./com4
f8: f1
        ./com5
```

a. Specify what files can cause com1 to be executed.

b. How could we only execute com4?

6a. Draw the finite state automaton for [a-z]?b?c+[a-z]*

b. Write a yacc grammar for the following pattern: $a^n bab^n$, $n \geq 0$

c. Give a simple example of a pattern that cannot be expressed in yacc.

7. Based on the talk about the problems of tcsh, explain the potential problem illustrated by each of the following:

a. if (! $variable) versus
if (! $variable)

b. grep "$var"'$'

c.

```
if ( $X ) then
   fgfdgfgfdfgfg
endif
```