Midterm Format (Tentative):

- Similar to previous quarters (see link on course web page to old exams)
  - Closed everything
- Unix
  - Shell commands (get around in the directory, manipulate files, etc.)
    - Write and/or explain various commands / options
  - Scripting (likely process something in a loop, use conditionals)
  - Write some regular expressions
  - Scripting (regular expressions and grep / sed)
- C
  - Write what a given C program outputs (pointers, pass-by-value, pass-by-reference)
  - Write a C program (likely string processing)

Midterm Topic List:

1. **Shell basic, commands, options**
   a. Directory commands (ls, cd, pwd, mkdir, rmdir)
   b. Directories (home, ~/, ., ..)
   c. Shell/system commands (man, info, clear, exit, date, cal, uname, echo)
   d. Getting help (man, info, -h, --h, -help, --help)
   e. File commands (cp, mv, touch, rm, chmod)
   f. File examination (cat, less, more, wc, diff)
   g. Searching and sorting (grep, sort, find, locate, which)
   h. Processes commands (ps, kill, killall, top)
2. **Processes, users, shell special characters, emacs.**
   a. Each process has private memory and I/O streams
   b. A running shell is just a process that kills itself when interpreting the exit command
   c. Command source
   d. Aliases (alias, unalias)
   e. .bash_rc, .bash_profile
   f. Emacs – a programmable, extensible text editor
3. **I/O Redirection, Standard Input/Output/Error, Pipes, Redirection, Quotes**
   a. Every process has three standard streams: stdin (0, input), stdout (1, output), stderr (2, error)
   b. Inputs and outputs of commands can be redirect using <,>
   c. Output (stdout) of one command can be "piped" to input (stdin) of another command using |
   d. Examples:
      i. Redirect input: cmd < file
      ii. Redirect output, overwrite file: cmd > file

iii. Redirect output, append to file: cmd >> file
iv. Redirect error output: cmd 2> file
v. Redirect output and error output to file: cmd &> file

e. Special characters – if want to use it in a file, needs to be escaped \
f. Quotes:
   i. "something" – threat something as a single argument, but allows substitution for $variables
   ii. 'something' – suppresses basically all substitutions and treats stuff literally

4. Shell Variables and Shell Scripts

a. Customizing the shell ($HOME, $USER, $PATH, $PS_1)
b. What if we want to run a bunch of commands w/o changing our shell's state? Answer: start a new shell (sharing our stdin, stdout, stderr), run the commands in it, and exit.
c. The script accesses the arguments with $i to get the ith one (name of program is $0).
d. $#, $n (where n is an integer), $@, $*, $?
e. Arithmetic:
   i. Variables are strings, so k=$i+$j is not addition.
   ii. But ((k=$i+$j)) is (and in fact the $ is optional here).
   iii. So is let k="$i + $j".
f. For/while loop, conditions
   i. file tests (-d, -f, -L, -r, -w, -x, -s, -nt, -ot)
   ii. string tests (=, !=, -z, -n)
   iii. numeric tests (-eq, -ne, -gt, -ge, -lt, -le)
   iv. logic (-a, -o, !, 0, 1)

5. Regular Expressions, grep, sed and other utilities

a. Globbing – refers to shell filename expansion
b. A regular expression *p* may "match" a string *s*.
   i. If *p* =
      1. –a, b, … matches the single character (basic reg. exp.)
      2. –*p*1*p*2, …, if we can write *s* as *s*1*s*2, where *p*1 matches *s*1, *p*2 matches *s*2.
      3. –*p*1 | *p*2, … if *p*1 matches *s* or *p*2 matches *s*
      4. (in egrep, for grep use \|)
      5. –*p*1*, if there is an $i \geq 0$ such that p1…p1 (*i* times) matches s.

c. Grep - prints any line that has one or more substrings that match .
d. Sed - For each line in file, replace every (longest) substring that matches *pattern* with *replacement* and then print it to standard out.
e. \(\) brackets
f. Option –r

6. **C: Control constructs, declarations, preprocessor, printf**
    a. Control constructs: while, if, for, break, continue, switch:
    b. Declaration vs Definition of Functions:
        i. ***Declaration*****:** introduces a name and describes its properties (type, # parameters, etc.), but does not create it
        ii. ***Definition*****:** the actual thing itself
    c. Preprocessors:
        i. Including contents of header files
        ii. Defining constants and parameterized macros (textual-replacements)
        iii. Conditional compilation
7. **Locals, types, left vs. right expressions**
    a. lvalues vs rvalues:
        i. Law #1: Left-expressions get evaluated to locations (addresses)
        ii. Law #2: Right-expressions get evaluated to values
        iii. Law #3: Values include numbers and pointers (addresses)
8. **Pointers, parameter passing by value, by reference**
    a. Address-space layout
    b. The call stack
        i. The call-stack has one “part” or “frame” (activation record) for each active function that has not yet returned. It holds:
            1. Room for local variables and parameters
            2. The *return address* (index into code for what to execute after the function is done)
            3. Other per-call data needed by the underlying implementation
9. **Tools: debugging – gdb**
    a. Commands:
        i. backtrace
        ii. list
        iii. break
        iv. run, step, next, continue, finish
        v. frame, up, down
        vi. print expression, info args, info locals