Name:_____

CSE374 Winter 2016, Midterm Examination
February 8, 2016

Please do not turn the page until the bell rings.

Rules:

- The exam is closed-book, closed-note, closed-calculator, and closed-electronics.
- **Please stop promptly at 1:20.**
- There are **72 points** total, distributed **unevenly** among **7** questions (many with multiple parts):

Question	Max	Earned
1	11	
2	14	
3	7	
4	9	
5	14	
6	9	
7	8	

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit. But clearly indicate your final answer.**
- The questions are not necessarily in order of difficulty. **Skip around.** Make sure you get to all the problems.
- There is reference material at the end of the exam.
- If you have questions, ask.
- Relax. You are here to learn.

Name:_____

1. (**11** points) (Shell commands) You have run a series of commands to explore your current directory. Note that `ls -F` just lists directories with a slash (/) at the end to distinguish them from regular files.

```
[you@midterm] pwd
/home/you
[you@midterm] ls -F
wav_to_mp3.sh  logs/  music/
[you@midterm] ls -F logs
progress.log  install.log   data.log  cache.log
boot.log      graphics.log
[you@midterm] ls -F music
albums/      bulletproof.mp3  cream_on_chrome.mp3  creep.wav
delilah.mp3  el_bawaba.wav    ... (many more music files)
```

Solve each problem *individually*. The starting state of the shell is always the same.

(a) You want to create a text file that lists each log file. From the current shell state, write additional commands to create a file `log_files.txt` containing the name of each `.log` file in the `logs` directory, one per line. `log_files.txt` should be in the logs directory.

(b) To save space, you want to delete the oldest log file. From the current shell state, write a one-line command that deletes the `.log` file in the `logs` directory with the oldest timestamp.

Name:_____

(c) Your music files are taking up too much space, so you want to convert your wav files to mp3 files (mp3 files use compression while wav files do not, so this will save space). From the current shell state, write additional commands to create an `.mp3` file from each `.wav` file in the `music` directory. You can ignore any files in subdirectories of `music`.

You should use the Bash script `wav_to_mp3.sh` to perform the conversion. The script takes two arguments. The first argument is the input wav file and the second argument is the output directory where the script will put the mp3 version of the input file (you can rely on the script to use an appropriate filename for the converted mp3 file). The script will also take care of removing the input wav file if the conversion is successful.

(d) You want to make a backup copy of your music directory. From the current shell state, write additional commands to create a new directory called `music_backup` and then copy the contents of the `music` directory into that directory.

Solution:

(a) `ls logs/*.log > logs/log_files.txt`

(b) `rm $(ls -t logs | tail -n 1)`

(c)
```
for file in $(ls music/*.wav); do
    ./wav_to_mp3.sh $file music
done
```

(d) `cp -r music music_backup`

Name:_____

2. (**14** points) (Bash scripting) Write a shell script `chooselines` that prints a number of random lines from each provided file. The command

   ```
   chooselines NUM FILE1 FILE2 ... FILEN
   ```

   should print out `NUM` different random lines from each of the files.

   To select and print out random lines from a single file, you should use the `choose` command. `man choose` provides the following documentation:

   ```
   NAME
       choose -- select random lines from a text file

   SYNOPSIS
       choose [--head] [--label=LABEL] [-n NUM] [-p PREFIX] [--tail] FILE

   DESCRIPTION
       The choose utility prints random lines from FILE. By default it selects a
       single line uniformly at random and prints it to standard output.

       The following options are available:

       --head
               Print lines starting from the top of FILE instead of randomly.

       --label
               Prefix printed lines with the name of FILE.

       -n NUM
               Print NUM random lines. No line will be selected more than once.

       -p prefix
               Only print lines that begin with the string PREFIX.

       --tail
               Print lines starting from the bottom of FILE instead of randomly.
   ```

   You script should meet the following specification:

   1. Print an informative error message to stderr if not enough arguments are provided.
   2. Print an informative error message to stderr for each file argument that does not exist.
   3. Print `NUM` different random lines from each file that does exist.

   Please write your script on the following page.

Name:_____

Write your answer to problem 2 here.

Solution:

```
#!/bin/bash

if [ $# -lt 2 ]; then
    echo "usage: chooselines NUM FILES..." 1>&2
fi

num=$1
shift # discard first argument, shift remaining args down

for file in $@; do
    if [ -f $file ]; then
        choose -n $num $file
    else
        echo "chooselines: $file does not exist" 1>&2
    fi
done
```

Name:_____

3. (**7** points) (regular expressions) Consider the following regular expression designed to match amounts of money given in dollars:

   ```
   \$[0-9]+
   ```

   In its current form, however, this expression fails to match some of the ways a dollar amount might be expressed. Please extend this regular expression to cover the following cases. Each extension must continue to match everything matched by previous versions (including the original, above).

   (a) Extend the regular expression above so that it also matches negative dollar amounts.

   (b) Extend the regular expression from part (a) so that it also matches dollar amounts that include cents. Note that to be a valid dollar amount, cents must be given in exactly two digits.

   (c) Extend the regular expression from part (b) so that it also matches dollar amounts that include commas when the amount is large enough (e.g., `$1,000,000.00`, `$34,567`, `-$999,999`, `$2,500.55`).

   Solution:

   (a) `-?\$[0-9]+`

   (b) `-?\$[0-9]+(\.[0-9]{2})?`

   (c) `-?\$[0-9]{1,3}(,[0-9]{3})*(\.[0-9]{2})?`

Name:_____

4. (**9** points) (`grep` and `sed`) Consider the command that prints the files in the current directory in *long listing format*

```
[you@midterm] ls -l
-rwxr-xr-x 1 you ugrad_uw  123630 Jan 22 11:55 clint.py*
-rw-r--r-- 1 you ugrad_uw   32672 Jan  9 15:44 img_a.png
-rw-r--r-- 1 you ugrad_uw  929717 Jan  1 18:58 img_a.txt
-rwxr-xr-x 1 you ugrad_uw   11488 Jan 25 12:06 line_count*
-rw-r--r-- 1 you ugrad_uw    1029 Jan 25 12:06 line_count.c
-rwxr-xr-x 1 you ugrad_uw    9872 Dec 28 17:09 moonlight*
-rwxr-xr-x 1 you ugrad_uw    9816 Dec 28 15:57 rosebud*
-rw-r--r-- 1 you ugrad_uw    4305 Feb  2 14:44 trie.c
-rw-r--r-- 1 you ugrad_uw    4209 Jan 31 19:51 trie.c~
-rw-r--r-- 1 you ugrad_uw     874 Feb  2 14:41 trie.h
-rw-r--r-- 1 you ugrad_uw     871 Jan 28 15:16 trie.h~
-rw-r--r-- 1 you ugrad_uw   14600 Feb  2 14:44 trie.o
```

From left to right, the columns give the following information for each file:
file permissions, number of links, owner name, owner group, file size (in bytes),
time of last modification, and filename

The whitespace between each column consists of one or more single spaces.

Give a one-line command (that includes `ls -l`) that prints to stdout the size of each file that was last modified in January. You can assume filenames do not contain spaces. For example, given the above output of `ls -l`, your command would print:

```
123630
32672
929717
11488
1029
4209
871
```

You may find `grep` and/or `sed` to be useful.

Solution:

```
ls -l | grep -E 'Jan ' | sed -r -e 's/^.*[ ]+([0-9]+) Jan .*$/\1/'
```

Name:_____

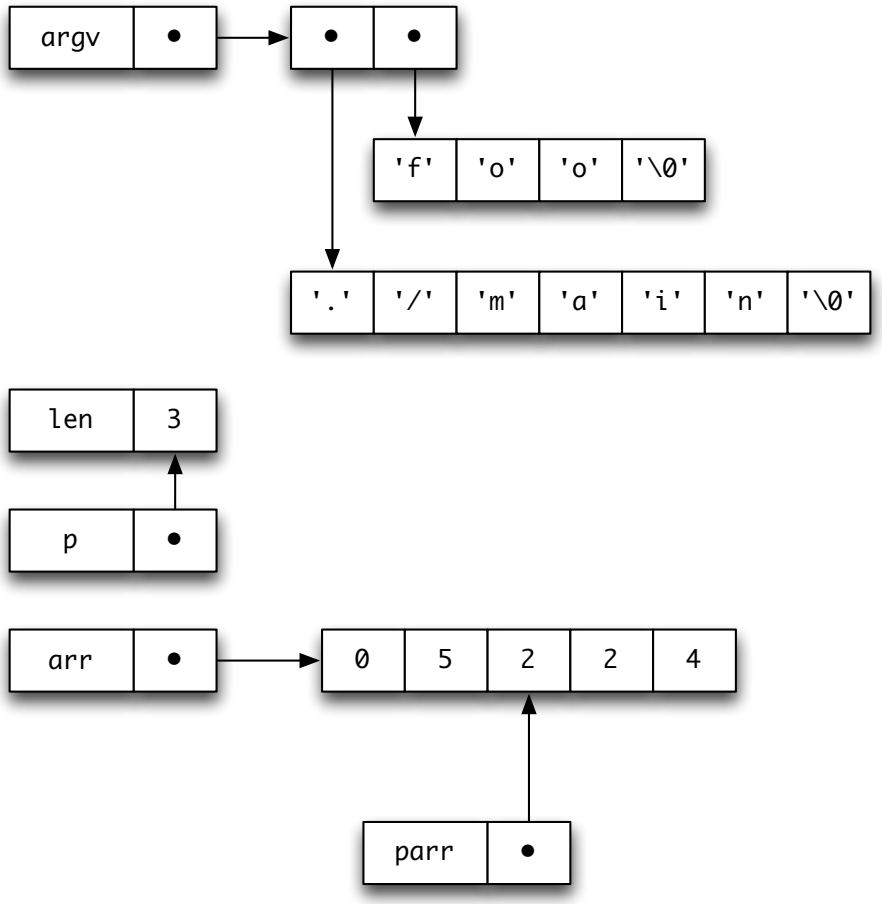
5. (**14** points) (C pointers) Consider the following C code in a file `main.c`:

```
#include <string.h>

int f(int *a, int *b) {
    *a = *a + *b;
    b = a;
}

int main(int argc, char **argv) {
    int len = strlen(argv[1]);
    int *p = &len;
    int arr[5];

    for (int i = 0; i < 5; i++) {
        *(arr + i) = i;
    }

    int *parr = arr;
    parr += 2;
    *(parr + 1) = *parr;

    f(parr - 1, arr + 4);
    return 0;
}
```

This code is compiled and then run as `./main foo`. Draw a picture of the arguments to `main` and the rest of the local variables just before `return 0;` executes in `main`. You don't need to consider how memory changes over time, just what it looks like at the end of the program. Use boxes and arrows to indicate data and pointers. Use a row of boxes to represent an array. Draw strings as they are actually represented in memory.

Name:_____

Solution:



Name:_____

6. (**9** points) (C structs) Your task is to define a `struct` to represent a telephone record. It should contain a name and a phone number, including area code. How many and what type of members you use are up to you, but you should use members that will be heap allocated where appropriate. You should also use `typedef` as part of the definition to provide a concise way to refer to the type.

   After you have defined the `struct`, you should write a function `match_name` that takes a pointer to telephone record and a string representing a name. The function should return an integer that indicates whether the telephone record matches the provided name. You can assume the telephone record is valid and accessing its members won't cause a segfault or other problems.

   You may find the C library function `strcmp` useful. Documentation for `strcmp` is given below:

```
int strcmp ( char *str1, char *str2 );

DESCRIPTION
    Compares the C string str1 to the C string str2.
    This function starts comparing the first character of each string. If they
    are equal to each other, it continues with the following pairs until the
    characters differ or until a terminating null-character is reached.

RETURN VALUE
    Returns an integral value indicating the relationship between the strings:
        < 0 means the first character that does not match has a lower value in str1
          0 means the contents of both strings are equal
        > 0 means the first character that does not match has a greater value in str1
```

   Write the definition of your struct and `match_name` below.

   Solution:

```
typedef struct {
    char *name;
    char *number;
} PhoneRecord;

int match_name(PhoneRecord *record, char *name) {
    return strcmp(record->name, name) == 0;
}
```

Name:_____

7. (**8** points) (memory model) Below is part of the implementation of a dynamically-sized array (similar to Java's ArrayList) of ints.

```
// creates and returns a new, empty vector with initial_len in initial storage
Vector* create_vector(int initial_len) {
    Vector *vec = malloc(sizeof(*vec));

    // data_len is the current total available storage
    // (i.e., maximum number of ints data can hold)
    vec->data_len = initial_len;

    vec->size = 0; // size is the number of ints in the vector

    // data is an array storing the ints in the vector
    vec->data = calloc(initial_len, sizeof(int));
    return vec;
}

// adds x to the end of vec
void add_int(Vector *vec, int x) {
    if (vec->size == vec->data_len) {
        int *new_data = calloc(vec->data_len * 2, sizeof(int));
        for(int i = 0; i < vec->size; i++) {
            new_data[i] = vec->data[i];
        }
        free(vec->data);
        vec->data = new_data;
        vec->data_len *= 2;
    }
    vec->data[vec->size] = x;
    vec->size++;
}

// returns the int at index i in vec
// caller must ensure i < vec->size
int get_index(Vector *vec, int i) {
    return vec->data[i];
}

// deallocates all memory used by vec
void destroy_vector(Vector *vec) {
    free(vec);
}
```

After you have read the above code carefully, please answer the questions on the next page.

Name:_____

(a) Describe what `add_int` is doing. You don't need to go over each line, but go into enough detail to explain how it is making `Vector` appear dynamically-sized (i.e., why the user of `Vector` never has to worry if there is enough space in the `Vector` to add another `int`).

(b) The current implementation of `Vector` contains a memory leak. Identify the source of the leak.

(c) Reimplement whatever functions from the previous page you deem necessary in order to fix this memory leak.

Solution:

(a) `add_int` is doubling the size of the interal array whenever adding another `int` would exceed the currently allocated storage. An array double the size of the current one is allocated, the existing data is copied over to it, the original array is freed, and the `Vector` is set to use the new array. This way, the `Vector` always behaves as if it has enough space.

(b) The internal array is not freed as part of `destroy_vector`, but the only pointer to it, `vec`, is freed.

(c)
```
void destroy_vector(Vector *vec) {
    free(vec->data);
    free(vec);
}
```

Name:_____

Reference

This is an incomplete list. **Just because a command or option is documented here doesn't mean there is a question that uses it.**

```
mkdir [OPTION]... DIRECTORY...
    Create the DIRECTORY(ies), if they do not already exist.
    -p, --parents
          no error if existing, make parent directories as needed
    -v, --verbose
          print a message for each created directory

ls [OPTION]... [FILE]...
    List information about the FILEs (the current directory by default). For each
    FILE that is not a directory, ls displays its name. For each FILE that is a
    directory, ls displays the names of files contained within that directory.
    Sort entries alphabetically unless -t is specifed.
    -R      Recursively list subdirectories encountered.
    -r      Reverse the order of the sort to get reverse lexicographical order
            or the oldest entries first.
    -t      Sort by time modified (most recently modified first) before sorting
            the operands by lexicographical order.

rm [OPTION]... FILE...
    rm removes each specified FILE. By default, it does not remove directories.
    -i    prompt before every removal
    -r, -R, --recursive
          remove directories and their contents recursively

cp [OPTION]... SOURCE DEST
or
cp [OPTION]... SOURCE... DIRECTORY
    Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
    -i, --interactive
          prompt before overwrite
    -R, -r, --recursive
          copy directories recursively

head [OPTION]... [FILE]...
    Print the first 10 lines of each FILE to standard output.
    -n, --lines=[-]K
          print  the first K lines instead of the first 10; with the
          leading '-', print all but the last K lines of each file

tail [OPTION]... [FILE]...
    Print  the  last  10  lines of each FILE to standard output.
    -n, --lines=K
          output the last K lines, instead of the last 10; or use -n +K to
          output starting with the Kth
```

Name:_____

```
cat [OPTION]... [FILE]...
    Concatenate files and print on the standard output.

echo [STRING]...
    Print the STRING(s) to standard output.

shell scripting
$(cmd)      substitute with the stdout from running cmd
$n          nth argument ($0 is the command itself)
$#          number of arguments (does not include $0)
$@          a list of all the arguments (does not include $0)
$?          the exit status of the most recent command
shift       discard the first argument ($1) and move the remaining arguments
            down ($2 becomes $1 and so on, this affects $# and $@).

for item in list_of_things
do
    ...
done

if TEST
then
    ...
fi

tests:
    [ -d file ]      true if file exists and is a directory
    [ -e file ]      true if file exists, regardless of type
    [ -f file ]      true if file exists, and is a regular file
    [ -n string ]    true if length of string is nonzero
    [ -z string ]    true if length of string is zero
    [ s1 = s2 ]      true if the strings s1 and s2 are identical.
    [ s1 != s2 ]     true if the strings s1 and s2 are not identical.
    [ n1 -eq n2 ]    true if integer n1 is equal to integer n2.
                     similarly for not equal (-ne), greater than (-gt),
                     and less than (-lt)

grep [OPTIONS] PATTERN [FILE...]
    grep searches the named input FILEs for lines containing a match to the given
    PATTERN. By default, grep prints the matching lines to standard out.
    -E, --extended-regexp
          Interpret PATTERN as an extended regular expression
    -i, --ignore-case
          Ignore  case  distinctions  in  both  the  PATTERN and the input files.
    -v, --invert-match
          Invert the sense of matching, to select non-matching lines.
    -x, --line-regexp
          Select only those matches that exactly match the whole line.
    -o, --only-matching
          Print only the matched (non-empty) parts of a matching line, with each
          such part on a separate output line.
```

Name:_____

```
sed [OPTIONS] COMMAND [FILE]...
    Applies COMMAND to each FILE line-by-line. By default sed prints each line of
    input to standard out.
    -e COMMAND      append the editing COMMAND to the list of commands
    -n              suppress printing of input lines to standard out

    Substitution commands have the form: s/SEARCH_PATTERN/REPLACEMENT/[modifiers]
    modifiers:
        p      print matched lines (necessary with -n)
        g      find multiple matches per line (default is first match only)

regular expressions
    [abc]      match one of the characters (a or b or c)
    [0-9]      match one of the chacters in a range
   [^ \t]      match one character not listed
        .      matches any character
    (xyz)      a group
       \n      match the nth group
        *      match 0 or more of the previous character or group
        +      match 1 or more of the previous character or group
        ?      match 0 or 1 of the previous character or group
      {n}      match the previous character or group exactly n times
    {n,m}      match the previous character or group at between n and m times
        ^      match the beginning of the line
        $      match the end of the line
```