Name: SOLUTIONS

Write your name in the space provided above.
Do not write your ID number or any other confidential information on this page.

Please wait to turn the page until everyone is told to begin.

While you are waiting, please read the following information:

- There are 6 questions on 12 pages worth a total of 100 points. Please budget your time so you get to all the questions. Keep your answers brief and to the point.
- Some question pages may be detached for your convenience (a separate page is provided for your answer in these cases). A stapler is available at the instructor podium if your entire exam falls apart.
- The exam is closed book, closed notes, closed electronics, closed Internet, closed neighbor, closed telepathy, etc., with the exception of the one instructor-provided double-sided page of notes.
- Many of the questions have short solutions, even if the question is somewhat long. Don't be alarmed.
- If you don't remember the exact syntax of some command or the format of a command's output, make the best attempt you can. We will make allowances when grading. Write legibly.
- Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

Score: _____ / 100

1. _____ / 16

2. _____ / 15

3. _____ / 17

4. _____ / 15

5. _____ / 17

6. _____ / 20

Question 1. (16 points) (Shell commands.) Suppose we have the following subdirectory structure inside the current working directory:

```
docs
docs/cse374
docs/cse374/notes.txt
docs/cse374/homework
docs/cse374/homework/main.c
docs/cse374/homework/package.c
docs/cse374/homework/package.h
docs/project
docs/project/deadlines.txt
docs/project/resources.txt
pictures
pictures/retouched.jpg
pictures/selfie.jpg
```

Write one or more `bash` commands to perform the following tasks. You should assume that each part of the question (A, B, C, etc.) is executed independently of the other parts, each part starts with the original set of files and directories as shown above, and each part starts in the initial current working directory.

(there may be equivalent sets of commands for some of these problems)

(A) Create a new subdirectory of `docs` named `txt`. Then write a single `mv` command to move all the txt files (files whose names end in `.txt`) shown above into that new subdirectory. Your `mv` command must use wildcards (`*.txt`) to specify the files – you may not write out the individual txt file names.

```
cd docs
mkdir txt
mv */*.txt txt
```

(B) Rename the `selfie.jpg` file to `original.jpg`.

```
cd pictures
mv selfie.jpg original.jpg
```

Question 1. (continued) Subdirectory structure repeated for your convenience:

```
docs
docs/cse374
docs/cse374/notes.txt
docs/cse374/homework
docs/cse374/homework/main.c
docs/cse374/homework/package.c
docs/cse374/homework/package.h
docs/project
docs/project/deadlines.txt
docs/project/resources.txt
pictures
pictures/retouched.jpg
pictures/selfie.jpg
```

Write `bash` commands to perform the following tasks. You should assume that each part of the question (A, B, C, etc.) is executed independently of the other parts, each part starts with the original set of files and directories as shown above, and each part starts in the initial current working directory.

(there may be equivalent sets of commands for some of these problems)

(C) Write a single command to remove the `cse374` directory and all of its contents.

```
rm –r cse374
```

(D) Create a new file named `temp.txt` in the `project` directory. Append the contents of both `deadlines.txt` and `resources.txt` into it in that order, and then print the number of lines in `temp.txt` .

(you could create the file `temp.txt` a number of ways)

```
cd docs/project
touch temp.txt
cat deadlines.txt > temp.txt
cat resources.txt >> temp.txt
wc –l temp.txt
```

Question 2. (15 points) (Regular expressions.)

Give regular expressions that could be used with `grep` (or `egrep`) that would match the lines described. (You do not need to indicate `grep` or `egrep` – as long as your answer works with one or the other you will receive credit.)

Assume we are searching a text file of vocabulary words where there is one word per line (similar to the `words` file we experimented with in class). Assume the entire file is in lowercase letters, although there may be some hyphenated words.

(A) You have a summer internship on the TV game show "Wheel of Fortune". To make the game harder for the contestants, you need to generate a list of words for the show that do **not** contain any of the most frequently occurring consonants in English vocabulary (`r`, `t`, `s`, `n`, `l`). (i.e., `academic` but not `stone` or `relations`)

similar to wi13 midterm, question 2 C; these expressions use `grep`

```
^[^rtsnl]*$
^[^rtsnl]\+$
^\([^rtsnl]\{1,\}\)$
-v [rtsnl]
```

(B) You are training for the Scrabble tournament in the next Olympics. You need to generate a list of words that contain **at least two** letters with the highest Scrabble point value (`q`, `z`, `j`, `x`, `k`). (i.e., `dizzy` and `joke` but not `zebra`)

```
.*[qzjxk].*[qzjxk].*
[qzjxk].*[qzjxk]
[q,z,j,x,k].*[q,z,j,x,k]
^.*[qzjxk].*[qzjxk].*$
```

(C) You put up the signs with moveable type at gas stations and other places. You accidentally ordered too many boxes of `a`'s, `b`'s, and `c`'s and need to use them up. Generate a list of words containing `a`, `b`, `c` in that order, possibly with other letters including `a`, `b`, `c` in-between. (i.e., `abacus` but not `bacon`)

similar to sp10 midterm, question 2 A

```
a.*b.*c
[a].*[b].*[c]
a.*b.*c.*
[a].*[b].*[c].*
.*a.*b.*c.*
.*[a].*[b].*[c].*
.*a\+.*b\+.*c\+.*
```

match substrings within a word, so these work.

```
^.*a.*b.*c.*$
^.*[a].*[b].*[c].*$
```

are anchored at both ends but have optional characters at the start and end of the word so they are not required to start with a and end with c; these work.

Question 3. (17 points) (Some `bash` scripting.)

Write a `bash` shell script that takes one or more arguments.

If there are no arguments, the script should print an appropriate usage message to `stderr` including the name of the script, and then `exit` with an exit status of `1`. Otherwise the exit status should be `0`.

For each argument,

- print "`processing `" and the name of the argument to `stdout`
- If an argument is not a regular file, then
  - print "`talk to the hand`" to `stdout`
- If an argument is a regular file and is empty (has 0 bytes in it), then
  - print "`nothing to see here`" to `stdout`
- If an argument is a regular file and is non-empty, then
  - print "`lines:` " and the number of lines in the file to `stdout`

(Your logic structure may vary from the outline above depending on how you organize your loop(s) and/or decision(s), how elaborate your test expressions are, and which commands you incorporate into your tests.)

Your script should work properly for any file name(s), including those with embedded spaces.

Hints (you may not need all of them):

- `if [[ -f` *`file`* `]]` tests *file* to see if it is a regular file.
- `if [[ -s` *`file`* `]]` tests *file* to see if it exists and its size is nonzero.
- `wc [`*`options`*`]` *`file(s)`* prints newline, word, and byte counts for each *file*, and a total line if more than one *file* is specified. Some options to select which counts are printed include `-l`, `-w`, `-b`.

Please write your answer on the next page.

You may detach this page from the exam if that is convenient.

Question 3 (continued). Write your shell script for Question 3 on this page.

```
#!/bin/bash

# exit if 0 arguments

# this portion similar to
# -- wi13 midterm question 4 (condition)
# -- wi12 midterm question 2 (redirection to stderr)
# $# is the number of arguments not including the script

if [[ $# -eq 0 ]]
then
    echo "$0: arguments required" >&2
    exit 1
fi

# this portion is similar to
# -- wi13 midterm question 3

while [[ $# -gt 0 ]]
do
    echo "processing $1"
    if [[ -f "$1" ]]
    then
          # must be a regular file
          if [[ -s "$1" ]]
          then
              # (regular and) nonempty
              echo `wc -l "$1"`
          else
              # (regular and empty)
              echo "nothing to see here"
          fi
    else
          # not a regular file
          echo "talk to the hand"
    fi
    shift
done
```

Question 4. (15 points) (Some more `bash` scripting.)

We have a text file with a list of names with one line per entry. Each line contains a "last name" (which can contain spaces), followed by a comma and a space, followed by a collection of one or more names separated by spaces that for convenience we will consider the "first name". Here are a few names from the file:

```
Cool, Joe I M
Bunny, Bugs
Mouse, Mickey The
Van Helsing, Abraham
```

We want to process these names to produce a list suitable for nametags. The new list should have each processed name on its own line, formatted as the first name followed by a space followed by the last name. On the above names, this result would look like:

```
Joe I M Cool
Bugs Bunny
Mickey The Mouse
Abraham Van Helsing
```

Write a `bash` shell script that has one argument giving the name of an input file containing data in the format described above. The script should read the input file, and write its output to `stdout`, but otherwise not modify the input file.

If the script has no argument, or more than one argument, it should print an appropriate usage message to `stderror`, and then `exit` with an exit status of `1`.

You may assume the data is "clean" in that every line is formatted as described above, and there are no extra commas in any line other than the one separating the last name and first name. You may also assume that if the script has an argument that it is a valid file name, the file exists, and it can be read.

Restriction: To process the input file, you must use `sed` and regular expressions.

Hint: `sed 's/`*`searchPattern`*`/`*`replacementPattern`*`/' `*`file`*

Please write your answer on the next page.

You may detach this page from the exam if that is convenient.

Question 4 (continued). Write your shell script for Question 4 on this page.

```
#!/bin/bash

# exit if not exactly 1 argument

# this portion similar to
# -- wi13 midterm question 4 (condition)
# -- wi12 midterm question 2 (redirection to stderr)
# $# is the number of arguments not including the script

if [[ $# -ne 1 ]]

then

     echo "$0: exactly one argument required" >&2

     exit 1

fi

# assume file exists, valid file name, can be read
# (thus no additional error checking)

# file formatted as lastName comma space firstName
# (.*) comma space (.*)
# find
# firstMatch comma space secondMatch
# replace with
# secondMatch space firstMatch

# use "" around the filename in case it has spaces

sed 's/\(.*\), \(.*\)/\2 \1/' "$1"
```

Question 5. (17 points) (C program output.) Consider the following C program:

```
#include <stdio.h>

void unknown(int *a, int *b, int *c) {
  *a = *b;
  *b = 42;
  *c = *a + *b;
}

int main() {
  int x;
  int y = 9;
  int z;
  int *q;
  int *r;
  int *s;

  q = &x;
  r = q;
  *r = 3;
  s = &y;
  z = 5;

  printf("XYZ: %d %d %d\n", x, y, z);
  printf("*: %d %d %d\n", *q, *r, *s);
  unknown(&x, &y, &z);
  printf("XYZ: %d %d %d\n", x, y, z);
  printf("*: %d %d %d\n", *q, *r, *s);
  unknown(q, r, s);
  printf("XYZ: %d %d %d\n", x, y, z);
  printf("*: %d %d %d\n", *q, *r, *s);
  return 0;
}
```

What output does this program produce when it is executed? (It does execute successfully.) It would be useful to draw diagrams showing variables and pointers to help answer the question and help us award partial credit if needed. If values change over time, cross out the old value and write the new value next to it rather than erasing the old value. Repeat as needed. (Hopefully this way we will be better able to follow your thought process if you get something incorrect.)

(see also the PDF of this problem worked out step by step)

```
XYZ: 3 9 5
*: 3 3 9
XYZ: 9 42 51
*: 9 9 42
XYZ: 42 84 51
*: 42 42 84
```

Question 6. (20 points) (C programming.)

Write a C program that reads a file whose name is given as an argument on the command line and prints the following on `stdout`:

- The name of the file
- Each line that is longer than 100 characters,
  - First in upper case
  - Then (on the next line) as it appeared originally
- The total number of lines in the file

Details:

- If the program is given no arguments or more than one argument, print a message to `stderr` and exit with a non-zero return code.
- If there is one argument, you can assume it names an existing file that can be successfully opened for reading.
- You can assume the input file contains lines of text, and no input line contains more than 250 characters, including any trailing newline ('`\n`') and binary zero terminator ('`\0`') characters.
- The program can be written as a single `main` function, although you may write other functions if you wish.
- Assume that any `#include` directives for any necessary C standard library functions are already provided.

Restrictions:

- You must read the input file as strings of characters, using one call to `fgets` to read each line.
- You must use appropriate string library functions to process each line.

Some (possibly useful) functions:

- `FILE * fopen(filename, "r")`
- `char * fgets(line, max_length, file)`
  - *returns null on end of file*
- `char * strncpy(dest, src, max_length)`
  - *append up to max_length characters from src to the end of dest*
- `int strlen(s)`
- `int toupper(ch)`
  - *uppercase character corresponding to ch, or ch if not a letter*

Please write your answer on the next page.
You may detach this page from the exam if that is convenient.

Question 6 (continued). Write your C program for Question 6 on this page.

(there may be variations on this solution)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_LENGTH 250

void printUpper(char *l) {
  int k;
  int len = strlen(l);
  char *lineCopy = (char *) malloc(len*sizeof(char));
  strncpy(lineCopy, l, len);
  for (k=0; k<len; k++) {
    lineCopy[k] = toupper(lineCopy[k]);
  }
  printf("%s\n", lineCopy);
  free(lineCopy);
}

int main(int argc, char *argv[]) {

  char line[MAX_LENGTH];
  FILE *fp = NULL;
  int numLines = 0;

  if (argc != 2) {
    fprintf(stderr, "one argument required\n");
    return 1;
  }

  fp = fopen(argv[1], "r");

  printf("%s\n", argv[1]);
  while (fgets(line, MAX_LENGTH, fp) != 0) {
    if (strlen(line) > 100) {
      printUpper(line);
      printf("%s\n", line);
    }
    numLines++;
  }
  printf("%d\n", numLines);

  return 0;
}
```