CSE 374 Midterm Exam Sample Solution 2/11/13

Question 1. (8 points) What does each of the following bash commands do?

(a) `rm x`

Remove the file named `x` in the current directory.

(b) `rm *x`

Remove all files in the current directory whose names end with the letter `x`. (Note: does not remove files whose names start with ".", but we did not consider that in grading the question.)

(c) `rm * x`

Remove all files in the current directory.

(d) `rm "* x"`

Remove the file named `* x` (three characters) in the current directory.

Question 2. (12 points) 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 your answer works with one or the other it will receive credit.)

(a) Lines containing the string `cse374`, but with any number of the letters capitalized (i.e., `cse374`, `CSE374`, `cSe374`, etc.)

```
[cC][sS][eE]374
```

(b) Lines containing the same digit repeated at least three times consecutively (i.e., 111, 222, 333, etc.).

```
\([0-9]\)\1\1
```

(c) Lines not containing any lower-case English vowels. (The vowels are the letters a, e, i, o, and u.)

```
^[^aeiou]*$
```

Question 3. (16 points) (A little scripting) Write a shell script that takes zero or more arguments. For each argument, if the argument is the name of an ordinary file that is empty (has 0 bytes in it), remove the file. If an argument is not an ordinary file name or if it is the name of an ordinary file but the file is not empty (has at least one byte in it), the script should ignore that argument and process any remaining arguments. If there are no arguments, the script should exit without doing anything.

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

Hint: `if [[ -f` *file* `]]` to test for a regular file.

```
#!/bin/bash

while [[ $# > 0 ]]
do
    if [[ -f "$1" ]]
    then
        size=`wc -c "$1" | sed 's/[^0-9]*\([0-9]*\).*/\1/'`
        if [[ $size -eq 0 ]]
        then
            rm "$1"
        fi
    fi
    shift
done
```

Besides using `sed` to isolate the number of bytes from the result returned by `wc`, either of the following would yield just the number, since here the input comes from `stdin`, which is anonymous:

```
cat "$1" | wc -c          wc -c < "$1"
```

Question 4. (14 points) A simple way to store an address book or telephone directory is in a text file with one line per entry. We have a file containing a collection of names, offices, and phone numbers. Here are a few sample lines from the file:

```
Ima EE Professor#EE B 735#3-1415
Fearless Leader#EE1 116#3-6515
E. Fudd#EEGAD 548#2-4784
B. Bunny#EE 037#254-5512
```

Each line contains the name of an individual, the office location (building and room number), and phone number. There is a single hash (#) character between each of the fields.

For some reason, the EE building has been denoted in several ways over the years: 'EE', 'EE B' (with one space between 'EE' and 'B'), 'EE1', 'EE 1' (with a space), and 'EEB'. We would like to use a shell script to create a clean copy of this file by replacing all references to the EE building with 'EEB'. For example, when run on the above input, the script should produce the following output:

```
Ima EE Professor#EEB 735#3-1415
Fearless Leader#EEB 116#3-6515
E. Fudd#EEGAD 548#2-4784
B. Bunny#EEB 037#254-5512
```

For this problem, write a shell script that has one argument giving the name of an input file in the format described above. The script should read the input file and copy it to standard output, changing all references to the EE building in the middle (office location) field to 'EEB' but otherwise not modifying the file.

If the script has no argument or more than one argument, it should print a suitable error message and exit without doing anything.

You may assume the data is clean: each line contains exactly three fields separated by '#' characters, there are no extra spaces before or after each '#' character, there is one space between the building name and room number, and so forth. You may also assume that if the script has an argument, it is a valid file name. You do not need to test for that.

Hint: `sed`

Restriction: You may not use PERL, Python, Ruby, AWK, or other languages to do the string processing.

Please write your answer on the next page.

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

Question 4 (cont.) Write your shell script here. (You almost certainly won't need all this space.)

```
#!/bin/bash

# exit if not exactly 1 argument
if [[ $# -ne 1 ]]
then
    echo exactly one argument required
    exit 1
fi

sed -E 's/#EE( ?[1B])? /#EEB /' "$1"
```

This solution used extended regular expressions for ease in using the ? operator. There are other solutions including brute force with multiple edit commands chained together.

Question 5. (16 points) Consider the following C program.

```
#include <stdio.h>

void mystery(int * a, int * b, int * c) {
  *a = *c;
  *b = *b * *a;
  *c = *a + *b;
}
int main() {
  int w = 5;
  int x = 1;
  int y = 3;
  int z = 2;
  mystery(&x, &y, &w);
  printf("%d %d %d %d\n", w, x, y, z);
  mystery(&w, &w, &z);
  printf("%d %d %d %d\n", w, x, y, z);
  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 to help us award partial credit if needed.

```
20 5 15 2

4 5 15 8
```

Question 6. (14 points) (Bugs 'R Us) One of your colleagues is having a terrible time getting a string function to work. He is trying to write a function that takes two strings and returns a new string that contains the first string followed by the second one. For example, the result of `string_plus("abc","wxyz")` should be a new string containing `"abcwxyz"` (with the appropriate '\0' terminator at the end, of course).

Here is the code. (If it is useful, the end of the question on the next page contains a summary of several functions, including these string functions.)

```
char * string_plus(char * str1, char * str2) {
  char * result;
  strcpy(result, str1);
  strcat(result, str2);
  return result;
}
```

(a) What is wrong with this code? Give a brief description of why it doesn't work. You should assume that both arguments are properly `'\0'`-terminated character strings – that's not the problem.

No space has been allocated to store the characters of the result string. (As a result, execution of both the `strcpy` and `strcat` function calls are very likely to cause a segfault.)

(b) Show how to fix the code so it works as specified. You should not alter the argument list or return type of the function. You can give the corrections to the function below, or show how to fix it by ***clearly*** showing the changes needed on the original code above. For full credit you must use string library functions to copy the original strings – you may not write loops to do it character-by-character. You may assume any necessary `#include`s are provided already – you do not need to write these.

Add the following line of code right before the call to `strcpy`:

```
result = (char*) malloc(strlen(str1)+strlen(str2)+1);
```

(Other solutions that have the same effect are fine. Allocating a local array and returning a pointer to it, however, is not since the result would be a dangling pointer to released space in the stack frame.)

Question 7. (20 points) (The small C programming exercise.) On the following page complete the definition of the function `files_match`, declared as follows:

```
int files_match(char * fname1, char * fname2);
```

The input parameters `fname1` and `fname2` are strings giving the names of two input files (for example, "story.txt", or "samples/datafile". The function should return the value 1 (i.e., true) if both files exist and they contain exactly the same lines of text. The function should return 0 (i.e., false) if either file cannot be opened, or if the files do not have the same contents.

Restrictions: Your function must read the files as strings of characters, using one call to `fgets` to read each line. It must use appropriate string library functions to compare or otherwise process the input lines – i.e., for full credit you may not write code to compare the files character-by-character.

To simplify things, you may assume the following:

- No input line in either file is longer than 200 characters, including the `'\0'` binary zero terminator at the end of each line.
- Any necessary `#include` directives for the standard C libraries are already provided – you do not need to write these.
- You do not need to close the input files before the function exits.
- If the input files can be opened, no errors will occur when reading from them.

Some (possibly) useful functions

- `FILE * fopen`(*filename*, "r")
- `char * fgets`(*line*, *max_length*, *file*), returns `NULL` on end of file
- `int    feof`(*file*), returns non-zero if end of *file* has been reached
- `char * strncpy`(*dest*, *src*, *max_length*)
- `char * strcpy`(*dest*, *src*)
- `char * strncat`(*dest*, *src*, *max_length*), append up to *max_length* characters from *src* to the end of *dest*.
- `char * strcat`(*dest*, *src*)
- `int    strncmp`(*string1*, *string2*, *max_length*)
- `int    strcmp`(*string1*, *string2*)
- `char * strstr`(*string*, *search_string*)
- `int    strlen`(*s*)

You may not need nearly as much space as is available for your answer.

Write your answer on the next page

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

Question 7. (cont). Complete the definition of the function below.

```
int files_match(char * fname1, char * fname2) {

  char line1[200], line2[200];
  char *p1, *p2;

  FILE * fd1 = fopen(fname1,"r");
  FILE * fd2 = fopen(fname2,"r");

  if (fd1 == NULL || fd2 == NULL)
    return 0;  // unable to open at least one file -
               //    return false

  while(1) {
    p1 = fgets(line1,200,fd1);
    p2 = fgets(line2,200,fd2);

    if (p1==NULL && p2==NULL)
      return 1;   // simultaneous eof - files match

    if (p1==NULL || p2==NULL)
      return 0;   // eof on one file but not the other -
                  //    mismatch

    if (strncmp(line1,line2,200) != 0)
      return 0;   // lines don't match, return false
  }
}
```