CSE 374 Midterm Exam Sample Solution 2/7/11

Question 1. (14 points) Below is a sequence of shell commands that we typed into a bash terminal window on a Linux machine and some of the output that was produced. There are several empty boxes. Your job is to write in each box the output produced by the previous command in the sequence.

The commands are executed consecutively and changes made by one command may affect the output produced by later commands. When the commands were run they all completed normally without producing any error messages.

Reference: the command `ls -F` produces the same output as a regular `ls` command, except that any directory name is followed by `/` to indicate that it is a directory.

```
> pwd
/home/me/test
> ls -F
hw1 hw1.bak tmp/
> bash
> touch it
> cd tmp
> ls -F
justme.txt
> echo "Dear Diary" > diary
> pwd
```

```
/home/me/test/tmp
```

```
> ls -F
```

```
diary  justme.txt
```

```
> exit
> pwd
```

```
/home/me/test
```

```
> ls -F
```

```
hw1  hw1.bak  it  tmp/
```

```
> ls -F *
```

```
hw1  hw1.bak  it

tmp:
diary  justme.txt
```

Question 2. (18 points) A typical system administration task is to create directories for new users. Rather than do it by hand, your boss has asked you to write a shell script to automate the process. The input to the script is a text file with user names. Example:

```
frodo
gandolf
```

Each name is on a separate line in the file and there are no blanks, tabs, or similar characters. Your script should create one new directory in the current directory for each user. Each created directory should have the same name as the corresponding user. For example, if the script is run in directory `/home` with the previous names in the input file, it should create `/home/frodo` and `/home/gandolf`. Do *not* assume that these are the actual names in the file or that the file contains any particular number of names. You can assume that the file contains at least one name if it makes any difference.

The permissions on each new directory should be set to `rwxrwxr-x`; i.e., read, write, and execute permission for the owner and group, and read and execute permission for others. The owner of each created directory should be changed to the user name. You can use the `chown` command for this. The command `chown` *who what* will change the owner of the file or directory named *what* to *who.* You should assume that the script will be run by someone with sufficient permissions to create the directories and change their owners and permissions.

Your script should check that it is called with one argument and that the argument is a regular file (`-f` option in a bash test command). You should also check that each new directory does not already exist (`-d`). If these problems occur, print a suitable message to `stderr` and exit with status 1. You do not need to check for any other errors. If you detect errors part way through the input file you do not need to back up and remove previously created directories.

write

your

answer

on the

next

page

------>

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

Question 2. (cont). Write your shell script here.

```
#!/bin/bash

# Check the number of arguments
if [[ $# -ne 1 ]]
then
   echo "Usage: ${0} nameList" >&2
   exit 1
fi

# Check to see if argument is a file
if [[ ! -f "$1" ]]
then
   echo "File ${1} does not exist or is not a regular file." >&2
   exit 1
fi

# Process user ids in file
usernames=`cat "$1"`
for i in $usernames
do
   # Check to make sure the directory does not exist
   if [[ -d $i ]]
   then
      echo "The directory ${i} already exists." >&2
      exit 1
   fi

   # Make directory
   mkdir $i

   # Change permissions
   chmod 775 $i

   # Change owner
   chown $i $i

done
```

There are many variations on these commands, which were fine. The `[ ]` test command can be used instead of `[[ ]]`. There are many ways to use `chmod` to get the rwxrwxr-x mode bits set properly. There are also various ways to read the names from the input file. Here is another common one:

```
while read i
do
done <"$1"
```

Question 3. (12 points) In the CSE Linux Virtual Machine, if we enter the command

```
rm x y
```

the `rm` command will ask if it is ok to delete each file:

```
rm: remove regular file 'x'?
rm: remove regular file 'y'?
```

That is not how the `rm` command behaves on a generic Linux system. Usually the files are removed without comment.

What happened is that the person who configured the CSE Linux VM made some sort of change so that whenever `rm` is executed it uses the `-i` option to request permission before deleting files. (i.e., when the user enters `rm`, the command `rm -i` is run instead).

Describe two distinct ways that this could have been done. Include enough technical detail so that your answer could be used to reproduce the same behavior on a generic Linux system. The original `rm` command executable file was not altered to do this, and your answer should assume that it remains unchanged.

a)

Include this alias command in the user's `.bashrc` or `.bash_profile` file when the account is created (this is what was actually done):

```
alias rm='rm -i'
```

b)

Create a short shell script named `rm` with the following contents, then put this script in some directory and alter the user's `$PATH` variable so that directory appears before the directory containing the regular `rm` command:

```
#!/bin/bash
/bin/rm -i "$@"
```

Question 4. (18 points) We have a file `phone-numbers` that contain names and phone numbers. One example is the following:

```
Uncle Sam         (202) 456-1414
Pocahontas  (617) 555-5555
Smith, John   (555) 111-3333
J. Q. Public, Jr. (312) 855-3754
```

As you can see, all of the phone numbers consist of a 3-digit area code in parentheses followed by a space followed by the 7-digit phone number with a dash in it. There are some blanks between the names and the area codes, but the amount of space is not always the same. Names consist of letters and punctuation like periods, commas, and spaces.

We would like to alter this file so the telephone numbers also include the international country code. For example, if the original number is `(206) 555-1212`, it should be changed to `001 1 (206) 555-1212`.

Give a `sed` command that will read the file `phone-numbers` and produce on standard output a version reformatted to include the country codes. You can assume that all phone numbers in the file are U.S. phone numbers and that `001 1` is the correct country code.

A fairly robust command that captures the phone number and replaces it with the same number preceded by the country code is

```
   sed 's/\(([0-9]+) [0-9]+-[0-9]+\)/001 1 \1/' phone-numbers
```

For this particular problem it's also ok to observe that since the character '(' cannot occur in a name, it's sufficient to match the left parentheses and insert the country code in front of it.

```
   sed 's/(/001 1 (/' phone-numbers
```

Question 5. (18 points) Consider the following C program.

```
#include <stdio.h>

void g(int *x, int *y) {
  *x = *y + 1;
  *y = *y + 2;
  printf("g: *x = %d, *y = %d\n", *x, *y);
}

void f(int *p, int *q) {
  int n = 17;
  g(q, &n);
  printf("f: *p = %d, *q = %d, n = %d\n", *p, *q, n);
}

int main(int argc, char** argv) {
  int i = 42;
  int j = 10;
  int n = 374;
  f(&i, &j);
  printf("main 1: i = %d, j = %d, n = %d\n", i, j, n);
  g(&n, &n);
  printf("main 2: i = %d, j = %d, n = %d\n", i, j, n);
  return 0;
}
```

What output does this program produce when it is executed? (It does execute successfully.) It would be useful to draw diagrams showing memory to help answer the question and to help us award partial credit if needed.

```
g: *x = 18, *y = 19
f: *p = 42, *q = 18, n = 19
main 1: i = 42, j = 18, n = 374
g: *x = 377, *y = 377
main 2: i = 42, j = 18, n = 377
```

Question 6. (20 points) (The small C programming exercise.) One of the summer interns wrote some C code that used a Linux function `strcasecmp`. This function does the same thing as `strcmp` (compares two strings), but it ignores the case of letters. The result of `strcasecmp(s,t)` is zero if `s` and `t` are the same ignoring case, a negative number if `s` is less than `t` ignoring case, and a positive number if `s` is greater than `t` ignoring case. The strings are compared using the underlying ASCII numeric codes of the characters except that corresponding upper case and lower case letters like '`a`' and '`A`' are treated as being the same. Some examples:

```
strcasecmp("abc", "ABC") == 0
strcasecmp("abc", "XYZ") < 0
strcasecmp("ABC", "xyz") < 0
strcasecmp("aBcdEF", "aBc") > 0
```

Unfortunately, `strcasecmp` is not a standard C library function and we need to run the program on a computer that only has a standard C implementation. Your job for this question is to implement `strcasecmp`.

Restrictions: Your code must work on any properly `\0`-terminated C strings of any length. It may not use arrays or dynamically allocate arrays to make copies of either string. It also may not alter either of the original strings.

Hints:

A couple of useful library functions:

```
int toupper(ch)  /* upper-case character corresponding to ch, or ch if not a letter. */
int tolower(ch)  /* lower-case character corresponding to ch, or ch if not a letter. */
```

Be sure your code works properly if the strings have different lengths.

Remember that C characters can be used as integer values for arithmetic and comparisons.

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 6. (cont). Write your answer here. You should assume that the file already contains `#include` statements for any necessary standard C libraries and you do not need to provide them.

```
/* return 0 if s==t, some negative number if s<t, or some */
/* positive number if s>t.  Upper- and lower-case letters */
/* that are otherwise the same are treated as identical.  */

int strcasecmp(char *s, char *t) {

   int cs, ct, diff;

   while (1) {

      /* convert chars as needed, and calculate difference */
      cs = tolower(*s);
      ct = tolower(*t);
      diff = cs - ct;

      /* if characters differ, diff is a suitable answer */
      /* (also handles case where one string is shorter) */
      if (diff != 0) {
         return diff;
      }

      /* if characters are same and are \0, strings match */
      if (cs == 0) {
         return 0;
      }

      /* advance to next characters */
      s++;
      t++;

   }

}
```

Note: most answers treated the strings as arrays and referenced characters using `s[k]` and `t[k]` instead of using pointer references `*s` and `*t`. Either was acceptable.