CSE 374 Midterm Exam Sample Solution 2/6/12

Question 1. (12 points) Suppose we have the following subdirectory structure inside the current directory:

```
docs
docs/friends
docs/friends/birthdays.txt
docs/friends/messages.txt
docs/cse374
docs/cse374/notes.txt
docs/cse374/hw9
docs/cse374/hw9/package.h
docs/cse374/hw9/package.c
docs/cse374/hw9/main.c
pictures
pictures/big.jpg
pictures/thumb.jpg
```

(a) What are the final contents of the file `answer` after the user executes the following commands in the current directory:

```
cd docs
ls friends > temp
ls cse374 >> temp
sort temp >> answer
```

birthdays.txt
hw9
messages.txt
notes.txt

(b) Assuming we are back in the original directory, and without changing the current directory, give a single command line that will list the number of lines and characters in each of the C source files (files with names ending in '`.c`') in the `hw9` directory. The output can contain additional detail or summary information if it is inconvenient to get rid of it.

`wc docs/cse374/hw9/*.c`

(Strictly speaking, we should use the -l and –c options to count just lines and characters, but that is not something you would be expected to memorize.)

(c) Assuming we are back in the original directory, and without changing the current directory, give a single command line that will make a complete copy of the `docs/cse374` subtree (including all directories and files), and store the copy in a new directory `docs/cse374-bak`.

`cp -r docs/cse374 docs/cse374-bak`

Question 2. (18 points) Write a shell script `rmdups` that deletes all files that are an exact byte-for-byte copy of a given file. The command

```
rmdups f1 f2 ... fn
```

should remove all of the files `f2` through `fn` that are exact copies of `f1`. Use `cmp` to detect whether two files are identical. The command

```
cmp -s x y
```

will exit with a return code of 0 if the two files `x` and `y` are identical. The return code will be 1 if they are different. The `-s` option instructs `cmp` to perform the comparison silently and not produce any output on `stdout` or `stderr`.

Your script should verify that there is at least one argument (`f1`) and that each of the files in the argument list exist and are regular files (`if [[ -f` *file* `]]`). If there are no arguments or if `f1` is not an existing regular file, the script should print an appropriate message to `stderr` and exit. If any of the other arguments are not existing regular files, the script should print an appropriate error message to `stderr` and ignore that argument, continuing on to process the remaining files.

Assume that whoever is running the script has sufficient permissions to remove any necessary files. Also assume that all files are readable if they exist. You do not need to worry about permissions issues.

Additional space is provided on the next page if you need more room for your answer.

Question 2. (cont.) Additional space for your answer if needed.

```
#!/bin/bash

if [[ $# < 1 ]]
then
    echo $0: at least one argument required >&2
    exit 1
fi

if [[ ! -f $1 ]]
then
    echo $0: $1 not ordinary file >&2
    exit 1
fi

fn=$1
shift
while [[ $# > 0 ]]
do
    if [[ -f $1 ]]
    then
        cmp -s $fn $1
        if [[ $? = 0 ]]
        then
           rm $1
        fi
    else
        echo $0: $1 not ordinary file >&2
    fi
    shift
done
```

The exact wording of error messages can be anything appropriate, but they did need to be written to `stderr` (stream 2).

Question 3. (14 points) Give **basic** regular expressions that could be used with `grep` (***not*** `egrep`) to locate words with the following properties in a file containing a list of words, each line in the file containing a single word (like the `words` file demonstrated in class).

(a) All words that would be legal identifiers in Java or C programs: that is, they start with an upper-case or lower-case letter and contain zero or more additional letters, digits, and underscore ('_') characters. Examples: 'x', 'xyzzy', 'a_long_name', 'Another_1', 'sillyQuestion', but not '1more_thing' (starts with a digit), etc.

```
^[a-zA-Z][a-zA-Z0-9_]*$
```

(b) All words that are 8 characters long and consist of the same 4 characters repeated twice. Examples: 'couscous', 'beriberi'.

```
^\(....\)\1$
```

(c) All words that do ***not*** contain any lower-case English vowels. For this question, the vowels are the characters 'a', 'e', 'i', 'o', and 'u'.

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

An answer that used + instead of * would also be fine.

On all of these problems, characters ^ and $ were needed surrounding each basic pattern to match only complete words. If those were missed on multiple parts of the question, points were only deducted once.

Question 4. (18 points) Several months before each quarter starts, faculty members are asked to send in a list of books for their upcoming courses. The bookstore only needs the 10-character ISBN codes (the codes in the barcode label on the back of each book), but usually faculty members send messages with extra information. For example:

> This year please get the Kernighan & Ritchie C book, isbn is 0-13-110362-8, 2nd ed
> Also include the Linux Pocket Guide 0-596-00628-0 by Barrett.
> For my other course please order Pragmatic Programmer 0-201-61622-X

For this problem, write a shell script whose argument is the name of a file containing lines like the ones above. The output should be a list of ISBN codes extracted from the text. For example, given the above input, the output should be:

> 0131103628
> 0596006280
> 020161622X

ISBN format: All ISBN codes in the input file contain 10 characters (mostly digits) plus 3 hyphens. There are 4 parts to the code separated by the 3 hyphens:

> *region-publisher-title-check*

The *region* is always 0, and the *check* character is a single digit 0-9 or the upper-case letter X. The *publisher* and *title* codes are all digits 0-9 and there are 8 digits total (you do not need to check for the exact number of digits in these parts). The number of digits in these two parts are different for different publishers. Large publishers have short publisher codes with fewer digits and more digits in the title code; small publishers have more digits in the publisher code and fewer digits for the title code. Both the publisher and title codes have at least one digit each.

You may assume that each line in the input file contains exactly one ISBN code somewhere in the line, and that nothing else in the line is formatted exactly like an ISBN code.

You should assume that the script has one argument and it is the name of a file that exists and can be read (i.e., you don't need to check for a valid argument list in this problem).

Hint: `sed`

```
#!/bin/bash
sed 's/.*0-\([0-9][0-9]*\)-\([0-9][0-9]*\)-\([0-9X]\).*/0\1\2\3/' $1
```

Note: Basic `sed` regular expressions do not include the + operator, but we allowed `[0-9]+` to match the book and publisher fields. Just `[0-9]*` would not be sufficient because it would match strings without digits in those positions.

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

```
#include <stdio.h>

void mystery(int *x, int *y, int *z) {
  *x = *y + *z;
}

int main(int argc, char** argv) {
  int x, y;
  int *p1;
  int *p2;
  int *p3;

  p1 = &x;
  *p1 = 1;
  p3 = p1;

  p2 = &y;
  *p2 = 3;

  printf("%d %d %d \n", *p1, *p2, *p3);

  *p3 = (*p2) + 1;

  printf("%d %d %d \n", *p1, *p2, *p3);

  mystery(p1,p2,p3);

  printf("%d %d %d \n", *p1, *p2, *p3);

  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.

```
1  3  1
4  3  4
7  3  7
```

Question 6. (20 points) (The small C programming exercise.) Write a on the next page a program that will read a file whose name is given as an argument on the command line and will print the following on `stdout`:

- The name of the file,
- The total number of lines in the file, and
- The contents of the last line in the file.

The output should be formatted as shown in the following examples.

Examples: If we run the program with the file `hamlet.txt` as input (which contains 4921 lines of text), the output should be:

```
hamlet.txt line 4921
FINIS. The tragedie of HAMLET, Prince of Denmarke.
```

If we run the program with a different file `fairy-tale.txt`, the output might be:

```
fairy-tale.txt line 263
And they all lived happily ever after.
```

Details:

- If the program is given no arguments or more than one argument, print a suitable message to `stderr` and exit.
- If there is one argument, you can assume it names an existing file that can be successfully opened for reading. You don't need to check for this.
- You can assume that the input file contains lines of text and no input line contains more than 200 characters, including any trailing newline (`\n`) and terminator (`\0`) characters.
- Your answer may be written as a single `main` function, but you can define additional functions if you wish.
- Assume that any `#include` directives for any necessary standard C library files are already provided – you do not need to write these.

Some useful functions: `fopen`(*filename*, "r"), `fgets`(*line*, *max_length*, *file*)

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.

```
#define MAX_LINE_SIZE 200

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

  int nlines;                /* number of lines read    */
  char line[MAX_LINE_SIZE];  /* contents of line nlines */
  FILE *in;                  /* input file              */

  /* exit if not exactly 1 filename argument */
  if (argc != 2) {
    fprintf(stderr, "exactly one argument required\n");
    return 1;
  }

  /* open file, read and count lines */
  in = fopen(argv[1], "r");
  nlines = 0;
  while (fgets(line, MAX_LINE_SIZE, in)) {
    nlines++;
  }

  /* print number of lines and last line in file */
  printf("%s line %d\n%s", argv[1], nlines, line);

  return 0;
}
```

Note: Solutions on an exam would not be expected to contain extensive comments or `#define` constants as shown above.