CSE 374 Midterm Exam 2/6/17 Sample Solution

Question 1. (12 points, 4 each) Suppose we have the following files and directories:

```
$ pwd
 /home/user
$ ls -l
-rw-r--r-- 1 user uw  10 Feb  4 15:49 combine.sh
drwxr-xr-x 2 user uw   2 Feb  4 15:51 hws
-rw-r--r-- 1 user uw  10 Feb  4 15:49 perform-measurement.sh
-rw-r--r-- 1 user uw 250 Feb  4 15:49 popular.html
-rw-r--r-- 1 user uw 100 Feb  4 15:49 popular.txt
drwxr-xr-x 2 user uw   2 Feb  4 15:50 prev-exams
-rw-r--r-- 1 user uw  10 Feb  4 15:49 results.txt
$ ls hws
hw1.sh hw2.sh hw3.txt hw4.c
$ ls prev-exams
sp-15.txt wi-16.txt sp-16.doc
```

In the following questions, use shell wildcards (*) and other patterns when you can. You may not write full file names unless absolutely necessary. Answer each of these questions individually, using the initial file and directory contents shown above. You should assume that `/home/user` is the current working directory at the beginning of each part of the question.

(a) The shell scripts in `/home/user` do not have execute permissions. Give a single Linux command that will add execute (`x`) permissions to the shell scripts in this directory (i.e., files whose names end in `.sh`).

```
chmod +x *.sh
```

(b) Give a single `wc` command that will list the number of characters, words, and lines for all files whose names end in `.txt` in this directory and both of its subdirectories.

```
wc *.txt */*.txt
```

(c) Give a Linux command to create a new directory `hw3` in `hws`, then use a second command to move the files (i.e., shell scripts, and text and html files) from `/home/user` to the new `hws/hw3` subdirectory. Be sure you don't accidentally move the subdirectories in `/home/user` to this new subdirectory.

```
mkdir hws/hw3
mv *.sh *.html *.txt hws/hw3
```

Notes: There are many other reasonable solutions to these questions, including things like selecting the "`popular`" files with "`pop*`". However, to receive full credit, solutions needed to use wildcards instead of full file names, and also use commands correctly.

Question 2. (20 points) (A little scripting) Write a shell script that will create backup copies of files and subdirectories in the current directory. Specifically, when the script is executed, it should do the following:

1. Create a directory named `.backup` in the current directory if it doesn't already exist. If it already exists you should use the existing directory with this name.
2. Copy all of the files and directories in the current directory into the `.backup` directory. Do not copy files or directories from the current directory whose names themselves begin with "." (i.e., files like `.chsrc` and directories like `.backup` itself).

If the script is unable to create the `.backup` directory, or if there is an ordinary (not directory) file in the current directory with that name, then the script should print an appropriate error message and exit with a return code of 1. (Use either `stdout` or `stderror` for the message.)

You do not need to remove any files that might be already present in `.backup`. Just copy files from the current directory to `.backup` and don't worry about its previous contents.

Hint: `mkdir`

Hint: The `cp` command has a `-r` option that will cause it to recursively copy any directories it encounters. You don't need to write any fancy control structures or use recursion in your shell script to copy all of the files and subdirectories found in the current directory.

Please write

your

answer

on the next page.

(You may remove this page for reference if you wish.)

Question 2. (cont.) Write your answer on this page.

```
#!/bin/bash

# Attempt to create .backup directory if there is no
# existing directory with that name.  mkdir will fail if
# .backup can't be created for some reason, including if
# there already is an ordinary file with that name.

if [ ! -d .backup ]
then
    mkdir .backup
    if [ $? -ne 0 ]
    then
      echo "could not create .backup directory"
      exit 1;
    fi
fi

# copy local files to .backup directory
cp -r * .backup
```

As usual, there are many possible solutions. For instance, instead of checking the `$?` result code after `mkdir`, another `if [ ! -d .backup ]` check could be used to determine if the `.backup` directory was successfully created. It could also be better to have an additional explicit check for an ordinary file named `.backup` and terminate with an error message if it exists without attempting to create a `.backup` directory, but a working solution was fine for an exam question.

Many solutions tried to use regular expression patterns like `'^[^.]*'` in the `cp` and other shell commands. That doesn't work – shell command patterns (globbing) are not the same as `sed`/`grep`/etc. regular expressions. The shell wildcard pattern `*` matches all names except those beginning with a `.`. The wildcard pattern `.*` matches all names that begin with a `.`. That is different from regular expressions, where `*` is an operator that applies to the previous regular expression.

Question 3. (16 points) (`sed` and `grep`) Suppose we have a text file named `staff.txt` containing staff member names and telephone numbers in this format:

```
Smith, John  (206)-555-5555
Doe,    Jane   (514)-222-2066
Wells,Mark   (425)-321-4567
Park, Crystal(206)-555-1212
Rodgers, Fred  (206)-333-3333
```

Give a single Linux command line that will print the first and last names (only) of all employees in file `staff.txt` whose telephone numbers have the area code 206. There should be a single space between the first and the last name. If `staff.txt` contains the data shown above, the output should be:

```
John Smith
Crystal Park
Fred Rodgers
```

You should not assume anything about the presence or absence of whitespace in the file, except that you can assume that the telephone numbers and the first and last names do not contain any embedded blanks or other whitespace.

Hints: your command line will probably have more than one command connected by the pipe (|) symbol. You should not assume that every occurrence of "206" in the staff.txt file is an area code – it might appear elsewhere in the phone number. You may assume that the parentheses characters '(' and ')' are only used to surround area codes.

Note: if the handwritten solution won't fit on one line, just continue on a second line at some obvious place, like right before or after any pipe (|) symbol.

Here are a few solutions:

```
grep '(206)' staff.txt | \
       sed 's/ *\([^ ]*\) *, *\([^ ]*\) *(.*/\2 \1/'
grep "(206)" staff.txt | \
       sed 's/\s*\([a-zA-Z]*\) *, *\([a-zA-Z]*\).*/\2 \1/'
```

Grading notes: The '\' character at the end of the first line is the standard Linux convention for continuing a command that won't fit on a single input line. That was not expected in any of the hand-written solutions.

These solutions handle any combinations of blanks in the input file, but they don't handle tabs. We should have been more precise to say that just skipping blanks would be sufficient. Also, since none of the examples showed any lines with leading blanks or blanks before the "," commas, we did not penalize solutions that did not handle those cases.

Question 4. (12 points) (C preprocessor) Recall that the C preprocessor reads a C program before the program is actually translated to executable code. The preprocessor modifies the program using the information in lines that start with a # character like `#define` and `#include`. Consider the following C code:

```
#define MAX 20
#define MIN 10

int foo() {
  int MAX = 25;
  int min = 15;
  int x, y;

  x = MAX + min;
  y = MIN + MINMAX;

  return x+y;
}
```

Below, show how the above code is transformed by the preprocessor by writing the modified version that results after all the # directives have been fully processed. This is the same as the output produced by `gcc -E` when the above code is given as input.

Hint: The preprocessor will successfully process this file. There are no guarantees, of course, that the resulting code will be a legal or correct C program. ☺

```
int foo() {
  int 20 = 25;
  int min = 15;
  int x, y;

  x = 20 + min;
  y = 10 + MINMAX;

  return x+y;
}
```

Notes: The preprocessor captures the information in `#define` directives and uses it to replace tokens in the file being processed, but does not copy the `#defines` to the output. It also does not simplify or rewrite the code further.

Question 5. (18 points) The traditional, annoying C program. As is usual, this program compiles and executes without warnings or errors.

```
#include <stdio.h>

void foo(int *a, int *b, int *c) {
  b[2] = -2;
  b[1] = a[0] + *c;
  *a = 15;
  printf("foo: *a = %d, *b = %d, *c = %d\n", *a, *b, *c);
}

int main() {
  int x = 42;
  int y[3];
  y[0] = 1;
  y[1] = 2;
  y[2] = 3;
  int * p = &y[1];
  p[1] = 17;

  printf("x = %d, *p = %d\n", x, *p);
  printf("y = {%d, %d, %d}\n", y[0], y[1], y[2]);

  foo(&x, y, p);

  printf("x = %d, *p = %d\n", x, *p);
  printf("y = {%d, %d, %d}\n", y[0], y[1], y[2]);
  return 0;
}
```

Fill in the lines below to show the output produced when this program is executed. Although not absolutely required, you should draw diagrams showing variables and pointers to help answer the question and to help us award partial credit if needed.
Output:

x = __42__, ***p** = __2__

y = { __1__, __2__, __17__ }

foo: *a = __15__, *b = __1__, *c = __44__

x = __15__, *p = __44__

y = { __1__, __44__, __-2__ }

Question 6. (22 points) (The small C programming exercise.) We would like to write a C program that opens a file named on the command line and copies that file to `stdout` replacing each single whitespace character in the file with a "." (period) character. The program should not add or remove any newline characters – those should appear in the output file exactly as they were in the input. However, every other whitespace character (including each individual blank or tab) should be replaced by a ".".

For this problem, a whitespace character is any character `c` for which the library function `isspace(c)` returns true (1).

Example: Suppose program argument (argv[1]) is `sample.txt` and the input file `sample.txt` contains

```
To be or not  2 be?
Is that a question?
```

then the program should copy this transformed version of `sample.txt` to `stdout`:

```
To.be.or.not..2.be?
Is.that.a.question?
```

Answer this question in the following two parts.

Grading notes: this solution assumes there is a `'\n'` at the end of the last line in the file, which is a reasonable assumption for an exam. For the same reason, the magic number "100" is used without `#defining` a constant, but in real code a named constant should be used instead. Also, since we didn't explicitly say that `isspace` returns true for all whitespace, including newlines (`'\n'`), we gave almost full credit to solutions that assumed it only returned true for tabs and blanks.

(a) (6 points) Complete the following function so that it replaces all of the whitespace characters in the string (`char` array) `s` with a `'.'` character. The number of elements (characters) in the array `s` is given by the parameter `len`. Assume that `len` has an appropriate value (i.e., `s` has at least that many elements). Also assume that any needed header files are already `#included` in the code.

```
/* Replace each whitespace char in s[0] through s[len-1]
   with a '.' (period) */
void replace_whitespace(char *s, int len) {
  for (int i = 0; i < len; i++) {
    if (isspace(s[i]))
      s[i] = '.';
  }
}
```

Question 6. (cont). (b) (16 points) Give an implementation of the program so that it opens the file whose name is given on the command line (`argv[1]`) and writes it to `stdout` with whitespace characters replaced by periods. If the program does not have exactly one argument on the command line (either the file name is missing or there are extra arguments), or if the file cannot be opened, the program should write an appropriate message to either `stdout` or `stderr` and terminate with an exit code of 1. If the program terminates successfully, it should have an exit code of 0.

The program should read one line at a time, and you may assume that no input line has more than 100 characters, including the newline `'\n'` and the `'\0'` byte at the end. Use the `replace_whitespace(s, len)` function defined in part (a) to replace whitespace characters with `'.'`. You may define additional functions if you need them, but you are not required to do so. Assume that all needed header files have already been `#include`d.

Hint: be sure not to clobber existing `'\n'` characters by replacing them with `'.'` and don't add extra newlines to the output. Remember that the `replace_whitespace` parameter `len` can have whatever value makes sense for what you are trying to do.

```
int main(int argc, char** argv) {
  char line[100];
  if (argc != 2) {
    printf("Wrong number of arguments\n");
    exit(1);
  }
  FILE * f = fopen(argv[1], "r");
  if (f == NULL) {
    printf("Unable to open file\n");
    exit(1);
  }
  // Strategy: read input lines and replace all but last
  // whitespace character (the '\n') with '.', then print.
  while(fgets(line, 100, f) != NULL) {
    replace_whitespace(line, strlen(line)-1);
    printf("%s", line);
  }
  return 0;
}
```