Project #3 Part 2 worth 5% [Project #3 is worth total of 15%]

Every one of the tasks should work on the Linux lab computers. Create a text file for your answers to Tasks 1 and 2. Tasks 3 and 4 provide shell scripts and place in a folder called PARTB. The PARTB folder should be compressed in your LASTNAME-PSUEMAILID-assign3.tgz file.

Task 1

Within the ./task1 directory are many nested directories and files. Find the files and corresponding line numbers within those files that have the word "hello" (lower case only). **Provide the command and results in your text file.**

Find the files and corresponding line numbers within those files that have a word (i.e., a string of alphabet characters) that starts with 'r' and ends with 's' (lower case only). **Provide the command and results in your text file.**

Task 2

Overall, your task is to create a static library, and it involves everything within the ./task2 folder. You'll note that there are three directories: ./lib, ./src, and ./include. **You cannot modify any of the given files.**

Within the ./src directory is a main.c file. Look inside and understand it. You can compile it without having to modify or add any files. The use of the term *compile* is very deliberate and specific here. Compiling, strictly speaking, does not mean create an executable. It means create an object file (main.o). **Write the command to do so in your text file.**

We need to link our main.o to a static library that provides the function definitions of all the secret_ functions. Within the ./include directory are three files: secret_math.h, secret_message.h, and secret_unused.h. Their corresponding .c files are non-existent and are what you are to create and compile into a static library. Go ahead and create the .c files for all the corresponding header files in the ./src directory. Their implementations are extremely trivial (see how main.c uses the functions). You have leeway in coding both the secret_message.c file and secret_unused.c file. In the former, just have the two functions print something. In the latter, it doesn't matter at all what you do. The point of the secret_unused.c file is to show you that static libraries can contain more compiled code than an individual might use. The "secret" prefix is to indicate to you that the internal implementations of these files is hidden once you compile and include them in a static library. Compile the three .c files in one command. **Write the command to do so in your text file.**

Now construct a static library named libsecret.a that contains all the compiled secret_ code. **Write the command to do so in your text file.**

Place the libsecret.a file in the ./lib directory. Now go back to the ./src directory and link the main.o with libsecret.a to produce an executable called final_program. **Write the command to do so in your text file.**

Run the program to make sure it works.

Task 3

You're to create a shell script (a bash script to be more specific) called **rename.sh**. It should rename all files of a certain type within the current directory to a new provided name concatenated with a number. For instance, given the following command:

```
./rename.sh jpg my_im
```

…then all the the .jpg files in the current directory will be renamed to:

```
my_im001.jpg
my_im002.jpg
my_im003.jpg [and so on…]
```

Thus, the first parameter is the file extension and the second parameter is the name prefix. If the user inputs more than two parameters, print out:

```
Illegal number of parameters
```

Complete in task3 folder and tar with project #3 code.

Task 4

You're to create another bash script. This time called **create_data.sh**, which is meant to create graph data to be used be as input to another program. For instance, given the following command:

```
./create_data.sh 3 2
```

A file called graph.data should be created within the same directory. Inside it might look like:

```
0 2
0 1
1 0
1 2
2 2
2 1
```

Each number represents a node in a graph. Each pairing of numbers on a line represents a directed edge from the left (source) node to the right (destination) node. Thus, in the above example, there are three nodes (0, 1, and 2), and there are 6 edges that connect the nodes.

The first argument to create_data.sh should be the number of nodes to create. The second argument is the number of edges each node should have.

The 2nd column (i.e., the destination of the directed edge) can be to any node (including the source node). However, the 2nd column must be random such that subsequent calls of the shell script produce different results. And for a given source node, it can only point to any destination node once.

Here's some more sample outputs:

For ./create_data.sh 2 1

```
0 1
1 1
```

For ./create_data.sh 4 2

```
0 1
0 0
1 2
1 0
2 1
2 3
3 3
3 0
```

If any of the inputs are 0, you must output:

Args can't be 0.

If the number of edges exceeds the number of nodes (say user provided 4 5 as arguments), you must output:

Number of edges (5) can't exceed number of nodes (4).

If the graph.data file already exists in the current directory you must output:

Do you want delete the old graph.txt and create a new one? [y or n]

And then if the user types "y", you remove the file and create a new one. If the user types "n", then you output "Then aborting." and do nothing.

Complete in task4 folder and tar with project #3 code.