System Programming in C – Homework Exercise 3

Publication date: *Thursday, May 7, 2020*
Due date: *Sunday, May 24, 2020* @ 21:00

General notes:

- A piped sequence of commands is an ordered sequence of Linux commands, each connected to its preceding command using the pipe symbol ('|'). Do not use any other way to combine commands (e.g. *process substitution, etc.*).
- You should only **use commands we saw in class**. For command **`grep`**, you may use the following options (if needed): **`-E -v -c`**. For command **`sed`**, use the **`-r`** option. **Do not use any other option for these two commands!**
- As always, make sure to use relative paths and not absolute paths. Your solution scripts should be executable from any location in the file system. In particular, the only absolute path you should use is `/share/ex_data/ex3/`.

Problem 1:

This problem involves additional parsing tasks for files `address_book.txt` and `story.txt`, which you parsed in Homework Exercise 2. Recall that `story.txt` contains chapters 8-10 of *The Great Gatsby* and that every line in the file `address_book.txt` has the following format:

<Name><dash><space>**<houseNumber>**<space>
<StreetName><space> **<StreetType>**<space>**<STATE>**

Where <dash> and <space> represent single characters ('-' and ' '), **<houseNumber>** is a positive integer in the range [1..9999], **<STATE>** consists of exactly two upper case letters, and all the other fields represent words that begin with an uppercase letter followed by at least one lower case letter.

1. Copy the files `address_book.txt` and `story.txt` from directory `/share/ex_data/ex3/` to the current directory.
2. Write a piped sequence of exactly two commands that takes the file `address_book.txt` and prints the lines corresponding to people whose first names have more than 4 letters. Lines should be printed according to lexicographical order of the first name into a file called `long_names.txt`.

Validation: file `long_names.txt` should contain exactly 7 lines, and the first two lines should be:

```
Andrew- 323 Judith Lane NY
Bruce- 7271 Canal Drive NJ
```

3. Write a piped sequence of commands that takes the file `address_book.txt` and prints a list of the first names of people, followed by a colon (:), a space, and the state, for all people who do not live in the states whose name starts with 'N' (e.g. NJ). Entries should be printed according to their original order in the file into file `non-N-states.txt`.

   Validation: file `non-N-states.txt` should contain exactly 5 lines. The first 2 lines are:
   ```
   Ryan: PA
   Peter: CA
   ```

4. Write a piped sequence of commands that takes the file `address_book.txt` and prints a list of people whose street type is "Street." For each such person you should print vethe following:

   ```
   <name> lives in the state of <state>, at this address: <number><space><Street Name> Street
   ```

   The 3 lines that result should be printed into file `live_in_street.txt`.

   Validation: file `live_in_street.txt` should contain exactly 3 lines, and the first line should be:
   ```
   Ryan lives in the state of PA, at this address: 4 Lanewood Street
   ```

5. Write a piped sequence of commands that prints the number of proper words in `story.txt` that contain at least 4 letters, one of which is the letter "u" (in upper or lower case). For this purpose, a proper word is defined as a contiguous sequence of alphabetical letters that: (1) immediately follows a white space or a punctuation mark, and (2) has a white space or a punctuation character immediately after it. For example, the following line contains exactly 5 proper words (the 5 proper words are underlined and highlighted in red):

   September 12th, 1906 . . . **I** **didn't** **work** 08.30-16.30

   Count the number of proper words that contain at least 4 letters, one of which is the letter "u", and print that number to the standard output. For example, for a file containing only the line above, you should print `0`. Note that each appearance of a word is counted toward the total count. For example, if the word `hunted` appears 5 times in `story.txt`, then it contributes 5 to the total count.

   Hint: you may find your solution to Problem 2.2 in Homework Exercise 2 useful, but note that the definition of 'word' there is not identical to 'proper word' as defined here.

   Validation: file `story.txt` contains exactly 812 words that contain at least 4 letters, one of which is the letter "u". They are all listed in their order of appearance in the story in `/share/ex_data/ex3/story-words-u.txt`.

6. Write a piped sequence of commands that prints the number of distinct proper words in `story.txt` that contain at least 4 vowels (a,e,i,o, or u). Proper words are defined as in (5) above. Note that each distinct word should be counted once, even if it appears multiple times in the story and in some of these cases, it contains upper case letters. Determine the number and print it as follows:

   ```
   The number of distinct proper words in story.txt that contain
   at least 4 vowels is <number>
   ```

   where `<number>` is the required number.

   Print this message by piping the piped sequence of commands that determines the word count into a **`sed`** command.

   Validation: file `story.txt` contains exactly 269 distinct words with at least 4 vowels. These words are listed in `/share/ex_data/ex3/story-4-vowels.txt` (in alphabetical order, in lower case, with their counts).

7. Write the commands you used in 1-6 in a single file named `ex3.1.sh`. The file should contain exactly 6 lines, each containing a single command or a piped sequence of commands.

Final validation and testing for Problem 1:

1. Verify that file `ex3.1.sh` has exactly 6 lines (use `wc`), and follow the specific validation steps specified for each item in 2-6.
2. Execute the commands in file `ex3.1.sh` as scripts (using `source`) and confirm that the following four files are created in the current directory (use `ls`):

   ```
   address_book.txt          live-in-street.txt
   long_names.txt            non-N-states.txt
   story.txt
   ```

3. The script should **print two lines to the standard output** corresponding to the output of 5-6. No error messages should be printed.
4. Run the script from **various locations** (not just `~/exercises/ex3/`) to ensure that it does not contain unwanted absolute paths.
5. Use the script `/share/ex_data/ex3/test_ex3.1` to test your solution. Execute the following command from directory `~/exercises/ex3/`:

   ```
   ==> /share/ex_data/ex3/test_ex3.1
   ```

   (the script produces a detailed error report to help you debug your code)

Problem 2:

1. Write a bash script named `list-items.sh` that receives as an argument a directory path and prints a list of all files or sub-directories in that directory. Follow these guidelines:
   - The script receives exactly two arguments: (1) a path for a directory and (2) a flag, which is either `-f` or `-d`.
   - If the first argument corresponds to a valid directory and the second argument is `-f`, the script lists all files in that directories (excluding sub-directories).
   - If the first argument corresponds to a valid directory and the second argument is `-d`, the script lists all sub-directories in that directories. The list should exclude files and directories which are not immediately below the given directory (see execution examples in the next page).
   - In both cases mentioned above, the script should exit with status 0.
   - If the script is not given exactly two input arguments, then the script should output the following message and exit with status 1:

     `Usage: <PATH-OF-SCRIPT-AS-CALLED> DIR_PATH -d|-f`

     where `<PATH-OF-SCRIPT-AS-CALLED>` is the path of script `list-items.sh` as called in the command line (hint: `$0`).
   - If the first argument does not correspond to a valid directory path, then the script should output the following message and exit with status code 2:

     `First argument <ARG> should correspond to an existing directory path`

     where `<ARG>` is the first input argument.
   - If the second argument is not `-f` or `-d`, then the script should output the following message and exit with status 2:

     `Second argument <ARG> should be -d or -f`

     where `<ARG>` is the second input argument.

Implementation guidelines:

- The script should be implemented as a stand-alone executable script. Make sure to have an accurate "`#!`" comment at the top of the file and grant the script executable permissions, to enable this.
- Use an appropriate conditional operator in an `if` statement to determine whether the first argument is a valid directory.
- To output a list of files or directories, use the `ls -l` command and pipe its output through a series of commands to filter items based on their ten-character permission string (recall shat is the difference between the permission string of a file and that of a directory)

Execution examples and validation:

```
==> ./list-items.sh /share/ex_data/ex3/testDir -f
item1
myFile.txt
==> echo $?      (← recall that this prints the exit status)
0
==> ./list-items.sh /share/ex_data/ex3/testDir -d
item2
stuff
==> echo $?
0
==> ./list-items.sh /share/ex_data/ex3/testDir/item2 -f
another-item
this_one_too.log
==> ./list-items.sh /share/ex_data/ex3/testDir/item2 -d
(prints nothing because there are no subdirectories)

==> ./list-items.sh
Usage: ./list-items.sh DIR_PATH -d|-f
==> echo $?
1
==> ./list-items.sh /share/ex_data/ex3/testDir -d -f
Usage: ./list-items.sh DIR_PATH  -d|-f
==> echo $?
1
==> ./list-items.sh /share/ex_data/ex3/testDir/item3 -d
First argument /share/ex_data/ex3/testDir/item3 should
correspond to an existing directory path
==> echo $?
2
==> ./list-items.sh /share/ex_data/ex3/testDir/item1 dud
First argument /share/ex_data/ex3/testDir/item1 should
correspond to an existing directory path
==> echo $?
2
==> ./list-items.sh /share/ex_data/ex3/testDir/item2 dud
Second argument dud should be -d or -f
==> echo $?
2
==> ./list-items.sh /share/ex_data/ex3/testDir -df
Second argument -df should be -d or -f
==> echo $?
2
```

You should try additional executions to validate your code. You should also use the script `/share/ex_data/ex3/test_ex3.2.1` to test your solution. This script checks your `list-items.sh` script using the execution examples above, as well as additional executions.

2. Write a bash script named `count-words.sh` that receives as an argument a path (for directory or file) and a word (string of non-space characters), and it prints the number of files mapped to the input path or its subdirectories that contain that word. Follow these guidelines:
   - The script receives exactly two arguments: (1) a path for a directory and (2) a query word, which is an arbitrary string of non-white-space characters.
   - If the first argument corresponds to a valid file path, then the script prints 1 if the file contains the query word at least one time, and 0 otherwise. The word is said to be contained in the file if it appears in the file in its exact form (case sensitive) and flanked on both sides by white spaces (or the beginning or end of the file). Special characters, such as punctuation marks are considered as any other character for this purpose.
   - If the first argument corresponds to a valid directory path, then the script outputs the number of files that: (1) contain the input word, as specified above, and (2) are contained in filesystem subtree rooted in the input directory (meaning that they are located in the input directory or one of its subdirectories, or one of its sub-subdirectories, etc.). See execution examples in the next page.
   - In both cases mentioned above, the script should print a single non-negative integer to the standard output and exit with status 0.
   - If the script is not given exactly two input arguments, then the script should output the following message and exit with status 11:
     `Usage: <PATH-OF-SCRIPT-AS-CALLED> PATH WORD`
     where `<PATH-OF-SCRIPT-AS-CALLED>` is the path of script `count-words.sh` as called in the command line (hint: `$0`).
   - If the first argument does not correspond to a valid path, then the script should output the following message and exit with status 202:
     `First argument <ARG> should correspond to an existing path`
     where `<ARG>` is the first input argument.
   - You may assume that the second argument, if given, is a valid word, meaning that it does not contain any white space characters.

Implementation guidelines:

- Follow the guidelines specified in Problem 2.1 to ensure that your script is implemented as a stand-alone executable script, and use appropriate conditional operators in an `if` statement to determine whether the first argument is a valid directory or file.
- If the first argument is a valid directory, then use appropriate calls to script `list-items.sh` from Problem 2.1 above to determine the files and directories included in that directory. For this purpose, you may assume that this script `list-items.sh` is located in the current directory. To examine files deeper in the subtree, the script `count-words.sh` should call itself in a recursive fashion. Use `$0` in your recursive call to make sure that you are calling the exact same script.

Execution examples and validation:

- review the contents of directory `/share/ex_data/ex3/testDir` and all files contained in its subtree to ensure that counts are correct.

```
==> ./count-words.sh /share/ex_data/ex3/testDir/myFile.txt hello
1
==> echo $?      (← recall that this prints the exit status)
0
==> ./count-words.sh /share/ex_data/ex3/testDir/item1 hello
0
==> echo $?
0
==> ./count-words.sh /share/ex_data/ex3/testDir/item1 Hello
1

==> ./count-words.sh /share/ex_data/ex3/testDir/stuff/moreStuff hello
0
==> ./count-words.sh /share/ex_data/ex3/testDir/stuff/ hello
0
==> ./count-words.sh /share/ex_data/ex3/testDir/item2 hello
2
==> ./count-words.sh /share/ex_data/ex3/testDir hello
3
==> ./count-words.sh /share/ex_data/ex3/testDir world
2
==> ./count-words.sh /share/ex_data/ex3/testDir you
4

==> ./count-words.sh
Usage: ./count-words.sh PATH WORD
==> echo $?
11
==> ./count-words.sh /share/ex_data/ex3/testDir hello world
Usage: ./count-words.sh PATH WORD
==> echo $?
11
==> ./count-words.sh /share/ex_data/ex3/testDir1 hello
First argument /share/ex_data/ex3testDir1 should correspond to
an existing path
==> echo $?
202
```

You should try additional executions to validate your code. You should also use the script `/share/ex_data/ex3/test_ex3.2.2` to test your solution. This script checks your `count-words.sh` script using the execution examples above, as well as additional executions.

Validation and testing for Problem 2:

1. Make sure to follow the implementation guidelines for Problems 2.1 and 2.2.
2. Use the script `/share/ex_data/ex3/test_ex3.2` to test your solution. This will run the individual tests for Problem 2.1 and 2.2.

   ```
   ==> /share/ex_data/ex3/test_ex3.2
   ```

   (the script produces a detailed error report to help you debug your code)

Submission Instructions:

1. After you validated and tested your solution, make sure that your `~/exercises/ex3/` directory contains the following scripts:
   - `ex3.1.sh`
   - `list-items.sh`
   - `count-words.sh`
2. your `~/exercises/ex3/` directory should also contain a `PARTNER` file with the user id of the non-submitting partner. The non-submitting partner should also add a `PARTNER` file containing the user id of the submitting partner.
3. Check your solution by running **`check_ex ex3`**. The script should be executed from the account of the submitting partner, and may be run from any directory. Clean execution of this script guarantees you 80% of the assignment's grade.
4. Once you are satisfied with your solution, you may submit it by running **`submit_ex ex3`**. The script should be executed from the account of the submitting partner, and may be run from any directory. You may modify your submission any time before the deadline (**24/5 @ 21:00**) by running **`submit_ex ex3 -o`** from any location.
5. For more information on the submission process, see the **Homework submission instructions** file on the course website.