System Programming in C – Homework Exercise 2

Publication date: *Thursday, April 23, 2020*
Due date: *Sunday, May 10, 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 weeks 1-2:

  `cp  cat  head  tail  cut  paste  wc  sort  uniq  tr`
- Do not use commands **`grep`** and **`sed`**, which we see in Lecture #3.
- 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, your scripts should not contain names of specific directories, other than the data source directory `/share/ex_data/ex2/`.

Problem 1:

1. Copy the file `address_book.txt` from directory `/share/ex_data/ex2/` to the current directory.

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. For example, the first three lines in the file are:

```
Andrew- 323 Judith Lane NY
Bruce- 8 Theatre Lane NJ
Ryan- 4 Lanewood Street PA
```

2. Write a piped sequence of commands that takes the file `address_book.txt` and generates a file `wrong_and_reformatted.txt` that contains the same number of lines as the original file, but with each line modified as follows:

   - Each digit in the address is replaced by the digit above it (for example, 4 is replaced by 5, and 5 is replaced by 6; note that 9 is replaced by 0).
   - Consecutive fields are separated by a single tab character (`'\t'`).

   Try to use a piped sequence with as few commands as you can. You can do this with as few as three commands (two pipe symbols).

Validation: file `wrong_and_reformatted.txt` should contain all the lines that are in the original file, in that same order. The first three lines should be:

```
Andrew  434  Judith    Lane      NY
Bruce   9    Theatre   Lane      NJ
Ryan    5    Lanewood  Street    PA
```

Note: The actual layout on the terminal might be slightly different depending on the size of the tabulated columns.

3. Write a piped sequence of commands that takes the file `address_book.txt` and generates a file `sorted.txt`. The new file has the same number of lines as the original file, but the lines should be sorted primarily by increasing numerical value of **`<houseNumber>`**, and secondarily by reverse lexicographic order of the composite string **`<StreetName>`**`<space>`**`<StreetType>`**. See example below.

Validation: file `sorted.txt` should contain the same number of lines as in the original file, but in a different order. The first five lines should be:

```
Peter- 4 Wildrose Road PA
Ryan- 4 Lanewood Street PA
Abe- 8 Theatre Lane PA
Bruce- 8 Theatre Avenue NJ
Jake- 8 Rockwell Drive NY
```

Note that Abe, Bruce, and Jake all share the same house number (`8`), so they are sorted in reverse lexicographical order of their street name (including the street type): “Theatre Lane” before “Theatre Avenue” before “Rockwell Drive.”

4. Write a piped sequence of commands that takes the file `address_book.txt` and generates a list of the four most frequent names in the file, sorted from the most frequent to the least frequent. The list should be appended to the end of file `sorted.txt`.

Validation: file `sorted.txt` should now contain an additional four lines:

```
      3 Peter
      2 Bruce
      2 Jake
      1 Abe
```

(number of spaces do not matter)

5. Write a piped sequence of commands that takes the file `address_book.txt` and generates a file `address_book_israeli.txt`. with the same format as the original, but with the **`<houseNumber>`** field after the street name and street type (which is the way addresses are written in Israel).

Validation: file `address_book_israeli.txt` should contain the same number of lines ordered as in the original file, and the first 3 lines should be:

```
Andrew- Judith Lane 323 NY
Bruce- Theatre Avenue 8 NJ
Ryan- Lanewood Street 4 PA
```

Hint: To implement this using a single piped sequence, you may wish to use this form of the command paste:

```
         ... |  paste address_book.txt - | ...
```

(We discuss the use of a dash (`-`) as an argument in Recitation #3 – #3 תרגול.)

6. Write the commands you used in 1-5 in a single file named `ex2.1.sh`. The file should contain exactly 5 lines, each containing a single command or a piped sequence of commands.

Final validation and testing for Problem 1:

1. Verify that file `ex2.1.sh` has exactly 5 lines (use `wc`) and follow the specific validation steps specified for each item in 2-5.
2. Execute the commands in file `ex2.1.sh` as scripts (using `source`) and confirm that the following four files are created in the current directory (use `ls`):
```
address_book_israeli.txt
address_book.txt
sorted.txt
wrong_and_reformatted.txt
```
3. The script should **not print anything to the screen** (neither errors nor standard output).
4. Run the script from **various locations** (not just `~/exercises/ex2/`) to ensure that it does not contain unwanted absolute paths.
5. Use the script `/share/ex_data/ex2/test_ex2.1` to test your solution. Execute the following command from directory `~/exercises/ex2/`:
```
==> /share/ex_data/ex2/test_ex2.1
```
(the script produces a detailed error report to help you debug your code)

Problem 2:

1. Copy the story file for chapters 8-10 of *The Great Gatsby* (called `story.txt`) from the shared directory `/share/ex_data/ex2/` to the current directory.

2. Write a piped sequence of commands that prints a list of all distinct words that appear in `story.txt` into the file `story-words.txt`. For this purpose, a word is defined as a non-empty contiguous sequence of characters that: (1) immediately follows a white space or a punctuation mark, (2) has a white space or a punctuation character immediately after it, and (3) does not contain a white space or a punctuation mark. Your count should be case insensitive, and each distinct word should be specified in lower case letters following its count. The list should be sorted alphabetically.

   For example, consider the following line of text:

   “`I, Great-Gatsby. . . feel great`”

   This line has four distinct words: “i”, “great”, “gatsby”, and “feel”. The word “great” appears twice, and the other three appear once.

   Validation: file `story-words.txt` should contain exactly 2046 lines. The expected file can be found in `/share/ex_data/ex2/story-words-expected.txt` and you may compare your version of the file to this expected version using the `diff` command (which we see in Lecture #3).

3. Write a piped sequence of commands that prints for each letter in the alphabet the number of distinct words that appear in `story.txt` and begin with that letter. You should assume the same definition of a word as in (2) above, and again, words should be considered in a case insensitive way (e.g., “Great” and “greAt” are the same word). Note that some words begin with a digit and not a letter. These words should not be considered in your count. For this purpose, you may assume that every digit 0-9 has at least one word that starts with it, and that digits precede letters in lexicographical order. Output this list into file `story-stats.txt`.

   Validation: file `story-stats.txt` should contain exactly 24 lines – one for each letter except X and Z. The letters should be specified in upper case (A, B, etc.), and in alphabetical order. The expected file can be found in `/share/ex_data/ex2/story-stats-expected.txt` and you may compare your version of the file to this expected version using the `diff` command (which we see in Lecture #3).

4. Write a piped sequence of commands that counts the number of times that the letter O appears in `story.txt` in lower and upper case (‘o’ and ‘O’). Print this number into file `story-num-os.txt`.

   Validation: file `story-num-os.txt` should contain exactly one line with a single number – 3106. You may check your piped sequence on other files that you create for validation.

5. Write the commands you used in 1-4 in a single file named `ex2.2.sh`. The file should contain exactly 4 lines, each containing <u>a single command</u> or <u>a piped sequence of commands</u>.

Final validation and testing for Problem 2:

1. Verify that file `ex2.2.sh` has exactly 4 lines (use `wc`) and follow the specific validation steps specified for each item in 2-4.
2. Execute the commands in file `ex2.2.sh` as scripts (using `source`) and confirm that the following four files are created in the current directory (use `ls`):

   ```
   story-num-os.txt
   story-stats.txt
   story.txt
   story-words.txt
   ```
3. The script should **not print anything to the screen** (errors or standard output).
4. Run the script from **various locations** (not just `~/exercises/ex2/`) to ensure that it does not contain unwanted absolute paths.
5. Use the script `/share/ex_data/ex2/test_ex2.2` to test your solution. Execute the following command from directory `~/exercises/ex2/`:

   ```
   ==> /share/ex_data/ex2/test_ex2.2
   ```

   (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/ex2/` directory contains the scripts `ex2.1.sh` and `ex2.2.sh`, and a `PARTNER` file containing 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. See **Homework submission instructions** for details.

2. Check your solution by running **`check_ex ex2`**. 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.

3. Once you are satisfied with your solution, you may submit it by running **`submit_ex ex2`**. 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 (**DATE 10/5 @ 21:00**) by running **`submit_ex ex2 -o`** from any location.

4. For more information on the submission process, see the **Homework submission instructions** file on the course website.