Exercise 41 - Name Sorter

Alphabetizing the contents of a file, or sorting its contents, is a great way to get comfortable manipulating a file in your program.

Create a program that reads in the following list of names from a file called `**exercise41_input.txt**` and sorts the list alphabetically:

```
Ling, Mai
Johnson, Jim
Zarnecki, Sabrina
Jones, Chris
Jones, Aaron
Swift, Geoffrey
Xiong, Fong
```

Then print the sorted list to a file called `**exercise41_output.txt**` that looks like the following example output.

Example Output

```
Total of 7 names
----------------
Johnson, Jim
Jones, Aaron
Jones, Chris
Ling, Mai
Swift, Geoffrey
Xiong, Fong
Zarnecki, Sabrina
```

Constraint

- Don't hard-code the number of names.

Challenges

- Implement this program by reading in the names from the user, one at a time, and printing out the sorted results to a file.
- Use the program to sort data from a large data set (e.g. census data) and use a profiler to analyze its performance.

Exercise 42 - Parsing a Data File

Sometimes data comes in as a structured format that you have to break down and turn into records so you can process them. CSV, or comma-separated values, is a common standard for doing this.

Construct a program that reads in the following data file (you will need to create this data file yourself; name it `**exercise42_input.txt**`):

```
Ling,Mai,55900
Johnson,Jim,56500
Jones,Aaron,46000
Jones,Chris,34500
```

```
Swift,Geoffrey,14200
Xiong,Fong,65000
Zarnecki,Sabrina,51500
```

Process the records and display the results formatted as a table, evenly spaced, as shown in the example output.

Example Output

```
Last      First     Salary
-------------------------
Ling      Mai       55900
Johnson   Jim       56500
Jones     Aaron     46000
Jones     Chris     34500
Swift     Geoffrey  14200
Xiong     Fong      65000
Zarnecki  Sabrina   51500
```

Constraints

- Write your own code to parse the data. Don't use a CSV parser.

Challenges

- Make each column one space longer than the longest value in the column.
- Format the salary as currency with dollar signs and commas.
- Sort the results by salary from highest to lowest.
- Rework your program to use a CSV parsing library and compare the results.

Exercise 43 - Website Generator

Programming languages can create files and folders too.

Create a program that generates a website skeleton with the following specifications:

- Prompt for the name of the site.
- Prompt for the author of the site.
- Ask if the user wants a folder for JavaScript files.
- Ask if the user wants a folder for CSS files.
- Generate an index.html file that contains the name of the site inside the <title> tag and the author in a <meta> tag.

Example Output

```
Site name: awesomeco
Author: Max Power
Do you want a folder for JavaScript? y
Do you want a folder for CSS? y
Created ./website/awesomeco
Created ./website/awesomeco/index.html
Created ./website/awesomeco/js/
Created ./website/awesomeco/css/
```

The user should then find these files and directories created in the working directory of your program.

Challenges

- Implement this in a scripting language on Windows, OSX, and Linux.
- Implement this as a web application that provides the specified site as a zip file.

Exercise 44 - Product Search

Pulling data from a file into a complex data structure makes parsing much simpler. Many programming languages support the JSON format, a popular way of representing data.

Create a program that takes a product name as input and retrieves the current price and quantity for that product. The product data is in a data file in the JSON format and looks like this (you will create this file as `**exercise44_input.json**`):

```
{
    "products" : [
        {"name": "Widget", "price": 25.00, "quantity": 5 },
        {"name": "Thing", "price": 15.00, "quantity": 5 },
        {"name": "Doodad", "price": 5.00, "quantity": 10 }
    ]
}
```

Print out the product name, price, and quantity if the product is found. If no product matches the search, state that no product was found and start over.

Example Output

```
What is the product name? iPad
Sorry, that product was not found in our inventory.
What is the product name? Galaxy
Sorry, that product was not found in our inventory.
What is the product name? Thing
Name: Thing
Price: 15.00
Quantity: 5
```

Constraints

- The file is in the JSON format. Use a JSON parser to pull the values out of the file (e.g. **https://github.com/google/gson (https://github.com/google/gson)**).
- If no record is found, prompt again.

Challenges

- Ensure that the product search is case insensitive.
- When a product is not found, ask if the product should be added. If yes, ask for the price and the quantity, and save it in the JSON file. Ensure the newly added product is immediately available for searching without restarting the program.

Exercise 45 - Word Finder

There will be times when you'll need to read in one file, modify it, and then write a modified version of that file to a new file.

Given an input file named **`exercise45_input.txt`,** read the file and look for all occurrences of the word *utilize*. Replace each occurrence with *use*. Write the modified file to a new file.

Example Output

Given the input file of

```
One should never utilize the word "utilize" in writing. Use "use" instead.
For example, "She uses an IDE to write her Java programs" instead of "She
utilizes an IDE to write her Java programs".
```

The program should generate

```
One should never use the word "use" in writing. Use "use" instead.
For example, "She uses an IDE to write her Java programs" instead of "She
uses an IDE to write her Java programs".
```

Constraints

- Prompt for the name of the output file.
- Write the output to a new file.

Challenges

- Modify the program to track the number of replacements and report that to the screen when the program finishes.
- Create a configuration file that maps “bad” words to “good” words and use this file instead of a hard-coded value.
- Modify the program so that it can handle every file in a folder of files instead of a single file.

Exercise 46 - Word Frequency Finder

Knowing how often a word appears in a sentence or block of text is helpful for creating word clouds and other types of word analysis. And it’s more useful when running it against lots of text.

Create a program that reads in a file named **`exercise46_input.txt`** and counts the frequency of words in the file. Then construct a histogram displaying the words and the frequency, and display the histogram to the screen.

Example Output

Given the text file `exercise46_input.txt` with this content

```
badger badger badger
badger mushroom
mushroom snake badger badger
badger
```

the program would produce the following output:

```
badger:   *******
mushroom: **
snake:    *
```

Constraint

- Ensure that the most used word is at the top of the report and the least used words are at the bottom.

Challenges

- Use a graphical program and generate bar graphs.
- Test the performance of your calculation by providing a very large input file, such as Shakespeare's Macbeth. Tweak your algorithm so that it performs the word counting as fast as possible.
- Write the program in another language and compare the processing times of the two implementations.