CSCE 156H/RAIK 184H Assignment 1 Spring 2015

Instructions Follow instructions *carefully*, failure to do so may result in points being deducted. You may discuss problems with your classmates, but all work must be your own. The CSE academic integrity policy is in effect (see https://cse.unl.edu/academic_integrity).

For each problem, you may be required to submit two non-trivial *test cases* which include a valid plain text input file and a plain text output file for each test case. The contents of these test cases are simply the input (command line arguments) and expected output. A test case is an input-output pair that is known to be correct—the solution should be worked out by hand or by use of a "known" correct mechanism. Simply using the output of your program as the valid output is *not* a test case. Points will be awarded based on the items outlined in the rubric.

Hand in your source files via the webhandin and be sure to verify that your programs work with the webgrader. Print out a copy of the rubric and hand it in hardcopy with your name and login on it.

For those with prior Java experience: your programs must be written in PHP, should accept command line arguments as specified, execute from the command line on CSE, and output properly formatted output to the standard output as specified.

For those without prior Java experience: your programs must be written in Java, should accept command line arguments as specified, execute from the command line on CSE, and output properly formatted output to the standard output.

1. **Program 1 – Natural vs Artificial Ordering**[1]

   Data has a "natural" ordering: numbers are ordered in nondecreasing order, strings are ordered lexicographically (according to the ASCII text table). However, in many situations, the natural ordering isn't the *expected* ordering. For example, class years are usually ordered Freshman, Sophomore, Junior, Senior whereas the natural ordering would order them Freshman, Junior, Senior, Sophomore.

   In this exercise, you'll sort a collection of strings according to an arbitrary artificial ordering. That is, instead of the A–Z alphabetic ordering, we will order them according to an alternative ordering of the English alphabet. As an *example*, one possible ordering would be:

   ```
   n e d c r h a l g k m z f w j o b v x q y i p u s t
   ```

   Your job will be to *sort* a list of English language words using a new ordering of the English alphabet.

   Specifically, you will read in an input file in the following format. The first line is the new ordering of English letters, each separated by a space. The second line is an integer

[1]Exercise inspired by http://www.reddit.com/r/dailyprogrammer/comments/20sjif/4192014_challenge_154_intermediate_gorellian/

indicating how many words are contained in the rest of the file. Each line after that contains a single English language word. You will read in this file, process it and produce a reordered list of the words sorted according to the new ordering.

An example input:

```
n e d c r h a l g k m z f w j o b v x q y i p u s t
vcawufotrb
laencfuesw
gvtkwekfom
vrsfqictqc
wmcvmjmtet
qetegyqelu
newaxdtjlt
nfrfrwkknj
fzqrvgblov
gkkmgwwwpa
```

```
Artificial Ordering:
newaxdtjlt
nfrfrwkknj
laencfuesw
gkkmgwwwpa
gvtkwekfom
fzqrvgblov
wmcvmjmtet
vcawufotrb
vrsfqictqc
qetegyqelu
```

For simplicity, you can assume that all words will be lower case and no non-alphabetic characters are used. However, you may not assume that all words will be the same length. Words of a shorter length that are a prefix of another word should be ordered first. For example, “newax” should come before “newaxn” in the ordering above.

- If using PHP, name your script `newOrder.php`; it should be invokable from the command line as:
  `php newOrder.php infile.txt` and output the results to the standard output.
- If using Java, name your source `NewOrder.java` and place it in the default package. Your program should be runnable from the command line as:
  `java NewOrder infile.txt` and output the results to the standard output.
- **Test Cases**: Design two test cases and hand them in. Make sure your test case files have the following names: `order_input_001.txt`, `order_output_001.txt`, and `order_input_002.txt`, `order_output_002.txt`.

2. **Program 2 – Ranked Voting**

   Ranked voting elections are elections where each voter *ranks* each candidate rather than just voting for a single candidate. If there are n candidates, then each voter will rank them 1 (best) through n (worst). Usually, the winner of such an election is determined by a *Condorcet* method (the candidate that would win in by a majority in all head-to-head contests). However, we'll use an alternative method, a *Borda count*.

   In a Borda count, points are awarded to each candidate for each ballot. For every number 1 ranking, a candidate receives n points, for every 2 ranking, a candidate gets $n - 1$ points, and so on. For a rank of n, the candidate only receives 1 point. The candidates are then ordered by their total points and the one with the highest point count wins the election. Such a system usually leads to a "consensus" candidate rather than one preferred by a majority.

   In this exercise, you will implement a Borda-count based ranked voting program. Your program will read in a file in the following format. The first line will contain an ordered list of candidates delimited by commas. Each line after that will represent a single ballot's ranking of the candidates and will contain comma delimited integers 1 through n. The order of the rankings will correspond to the order of the candidates on the first line.

   Your program will take an input file name as a command line argument, open the file and process it. It will then report the results including the point total for each candidate (in order) as well as the overall winner. It will also report the total number of ballots. You may assume each ballot is valid and all rankings are provided.

   An example input:

```
Alice,Bob,Charlie,Deb
2,1,4,3
3,4,2,1
4,2,3,1
3,2,1,4
3,1,4,2
```

   An example output:

```
Election Results
Number of ballots: 5

Candidate  Points
Bob        15
Deb        14
Charlie    11
Alice      10

```

```
Winner is Bob
```

- If using PHP, name your script `vote.php`; it should be invokable from the command line as:
`php vote.php infile.txt` and output the results to the standard output.
- If using Java, name your source `Vote.java` and place it in the default package. Your program should be runnable from the command line as:
`java Vote infile.txt` and output the results to the standard output.
- **Test Cases**: Design two test cases and hand them in. Make sure your test case files have the following names: `vote_input_001.txt`, `vote_output_001.txt`, and `vote_input_002.txt`, `vote_output_002.txt`.

3. **Program 3 – Isotope Halflife**: The rate of decay of a radioactive isotope is given in terms of its half-life H, the time lapse required for the isotope to decay to one-half of its original mass. For example, the isotope Strontium-90 (^{90}Sr) has a half-life of 28.9 years. If we start with 10kg of Strontium-90 in January 2009, then by the end of October 2037 we would expect to have only 5kg of Strontium-90 (and 5kg of Yttrium-90 and Zirconium-90, isotopes which Strontium-90 decays into).

Write a program that takes the following input:

- Atomic Number (integer)
- Element Name
- Element Symbol
- H (half-life of the element)
- m, an initial mass in grams

Your program will then produce a table detailing the amount of the element that remains after each year until less than 50% of the original amount remains. This amount can be computed using the following formula:

$$r = m \times C^{(y/H)}$$

where $C = e^{-0.693}$, y is the number of years elapsed, and H is the half-life of the isotope in years.

For example, using your program on Strontium-90 (symbol: Sr, atomic number: 38) with a half-life of 28.9 years and an initial amount of 10 grams would produce a table something like:

```
 Strontium-90 (38-Sr)
Elapsed Years    Amount
--------------------------
-                10g
```

```
1                   9.76g
2                   9.53g
3                   9.30g
...
28                  5.11g
29                  4.99g
```

- If using PHP, name your script `halfLife.php`; it should be invokable from the command line as:
  `php halfLife.php 38 Strontium-90 Sr 28.9 10` and output the results to the standard output.
- If using Java, name your source `HalfLife.java` and place it in the default package. Your program should be runnable from the command line as:
  `java HalfLife 38 Strontium-90 Sr 28.9 10` and output the results to the standard output.
- **Test Cases**: Design two test cases and hand them in. Make sure your test case files have the following names: `halflife_input_001.txt`, `halflife_output_001.txt`, and `halflife_input_002.txt, halflife_output_002.txt`.

4. **Honors**

   You will write an HTML webform for the half life program (place it in a file named `isotope.html` which will take several text inputs (for each of the inputs, and through a submit button, post the given text to a PHP script (name it `isotopeForm.php`) which will send an HTML formatted response to the user with the table displayed in a nicely formatted HTML table. The styling details are left up to you, but the form and the response should be well-formatted HTML that clearly communicates the output to the user.

   If you are doing the assignment in Java, you may instead write a Java program that outputs the raw HTML; place your source in a file named `IsotopeHtml`.