CS 106 Winter 2019

Lab 09: Text Processing

Due: Wednesday, March 20th, 11:59pm

SAVE the sketch as “L09Q01”.

Starter code: L09StarterCode.zip

Question One: Word Play

In this exercise, you will practice processing text by solving word puzzles. These puzzles involve iterating over an array containing all the words in some long list, finding the ones that pass some sort of criterion or test. You will be given a sketch that has a user interface for displaying the solutions to these puzzles; your job is to write the tests themselves.

1. Download the starter code and unzip it. Open the `L09Q01` sketch. You will see that the sketch is divided into two tabs. The `L09Q01` tab sets up the sketch, loads in the word list, and builds the user interface. You do not need to change anything in that tab, apart from adding your name and student ID to the top. Everything else you do will happen in the `Puzzles` tab.
2. There are seven puzzles to solve. They are solved by calling the functions `getWords_0()` ... `getWords_6()`. Each function must return an array of `Strings` containing all the words that solve the puzzle. You do not need to call these six functions yourself—they are called for you by the code in the `L09Q01` tab. You will find that `getWords_0()` is solved for you as an example. You must add code to the other six functions to solve those puzzles (look for comments marked `TODO`).
3. In `getWords_1()`, write code to find all words that start with "und" and end with "und". Example: "underground".
4. In `getWords_2()`, write code to find all words that contain all the vowels (a, e, i, o, u) at least once.
5. In `getWords_3()`, write code to find all words of six or more letters in which the letters are in strict alphabetical order within the word, with no repeats. Example: "almost", because "a" comes before "l", "l" comes before "m", and so on. You'll need an inner loop, here comparing each letter to the next one in the word. Note that you can use `<` and `>` on two `char` values to determine whether one comes before or after the other in the alphabet.
6. In `getWords_4()`, write code to find all words of 14 or more letters in which no single letter occurs more than once anywhere in the word. So, for example, "undiscoverable" doesn't count because it contains two "e"s, but "undiscoverably" words.

There are a couple of different ways to solve this puzzle. The easiest is to first write a helper function that counts the number of times a given letter occurs in a word. Here is the code for that.

```
int countLetter( String str, char ch )
{
  int total = 0;

  for (int idx = 0; idx < str.length(); idx++) {
    if (str.charAt( idx ) == ch) {
      total++;
    }
  }
  return total;
}
```

Now, in `getWords_4()`, first check that the current word has at least 14 letters. If so, check how many times each letter of the alphabet occurs in the word, by calling `countLetter()`. If any letter returns a count of two or more, this word is invalid and should not be appended to the output array.

7. In `getWords_5()`, write code to find all words with exactly eight letters where, if you take the first two letters of the word and move them to the end, you get another valid word. For example, if we move the "by" in "bypasser" to the end of the word we get "passerby", which is also in our word list. It will be useful to remember how to use the `String` class's `substring()` method.

   Let's say you're given the word "pancakes". Shifting two letters gives you "ncakespa". You now need to check whether that's a word. *Do not* loop over the array `words` looking for "ncakespa". Instead, use the variable `word_dict`, defined in the `WordPlay` tab: specifically, check if it contains the test word as a key, as shown in class.

8. In `getWords_6()`, write code to find all words in which the vowels "a", "e", "i", "o", "u" and "y" appear in the word in that order, with no other vowels. Example: "facetiously". The easiest way to solve this puzzle is with a regular expression. The good news is that the regular expression is provided for you! All you need to do is use it correctly. Use the built-in `match()` function with the current word and the regular expression. If `match()` returns a non-`null` value, then the pattern was found and the word is one of the solutions to the puzzle.

In case it's not obvious, *your code must actually find the words that solve the puzzle*. That is, you're not allowed to determine the solution words by some other means and return them explicitly in an array. Looked at another way, if we changed the file `words.txt`, your code would proceed to find new sets of solutions relative to the words in that new file.

Save your work in a sketch titled `WordPlay`.

Submission

Submit all sketch directory from this lab as one ZIP file called L09.zip to the lab dropbox on Learn.

It is your responsibility to submit to the correct dropbox with the correct files before the deadline. Otherwise you will receive a mark of 0.