CSC 360 Programming Assignment #3: Recursion
Due Date: Monday, September 15

In this assignment, you will implement a single program (class) which will consist of one main method, several recursive methods, and any helper methods that you require. Since all of the methods will be in the same class as main, you will have to make all of your methods static (this will not impact the performance of your recursive methods). There are four different String-based operations to implement. Each of these will require one recursive method apiece and possible helper methods such as for permutations a method to obtain the substring of s without character c.

The operations are as follows:

1. Permutations – derive all of the permutations of the given String. The permutations are all of the possible orderings of the characters of the String. We will assume that the String has no duplicate characters (for instance, we would not use "abbc" as a String). A String of n characters has n! permutations. The String "abcd" has the following 24 permutations:

abcd	abdc	acbd	acdb	adbc	adcb
bacd	bacd	bcad	bcda	bdac	bdca
cabd	cadb	cbad	cbda	cdab	cdba
dabc	dacb	dbac	dbca	dcab	dcba

To solve this problem, we need to use an ArrayList to store all of the permutations created. We want to ensure that a String does not already exist in this ArrayList before adding it, so for instance if our ArrayList variable is called list and we have a new permutation (String) called temp, we might use code like this:

```
if(!list.contains(temp)) list.add(temp);
```

The permutation method will return an ArrayList<String> and operate as follows.

permutations(String s)
- Create an empty ArrayList<String>, we will call it list
- if s is "" return an ArrayList list with "" as the only String
- else for each character, c, in the s
  - obtain the substring, rest, of s without c (if s is "abcd" c is 'c' then rest is "abd")
  - create a temporary ArrayList<String>, perms=permutations(rest)
  - for each String, s2, in perms, create two new Strings, c+s2 and s2+c and add these Strings to list if they are not already in list
  - return list

2. Alphabetical pairs – pass a String to a recursive method which will determine if the String contains pairs of letters at opposite ends of the alphabet, returning true or false. For instance, the word "wizard" has this property because 'a' and 'z' are opposite ends, 'd' and 'w' are at opposite ends and 'i' and 'r' are at opposite ends. *Opposite* means that the distance between a letter and 'a' is the same as another letter's distance from 'z'. For 'd' and 'w', these are both 3 positions away from 'a' and 'z' respectively. NOTE: you may assume lower case letters only. This recursive method works as follows:
   - If the String length is 0 return true (we will assume only words with an even number of letters)
   - Else select the 0^{th} character of the String and compute its distance from 'a', let's call this distance d1. For the rest of the String, find the first character which is d1 distance from 'z'. If not found, return false, otherwise remove the two letters from the String (the 0^{th} character and its pair found somewhere else in the String) and then return what you get by recursively calling this method with this reduced substring.

As an example, we start with "wizard". Length is not 0, do the else clause. Our 0^{th} character is 'w' which is 'w'-'a' distance from 'a', or 119-97=22 (119 is the ASCII value for 'w', 97 is the ASCII value for 'a'). We now search for a character in "izard" that is 22 away from 'z'. We find this in 'd'. We remove 'w' and 'd' leaving us with "izar" and then recursively call the method with this substring. Eventually we recursively call the method will a String of length 0 which returns true so the other method calls return true.

3. Recursive alphabetic merge – merge two alphabetized Strings, returning the merged String. For instance, the two Strings "aaaccff" and "bcddeg" will be merged into "aaabcccdeffg". In the case of merging 'c' and 'c', it doesn't matter if you select the 'c' from the first String or the second String. The recursive relationship looks like this:
   Base case: have we reached the end of one String? If so, return the other String (we test for end of String by comparing the String to "")
   Recursive cases: if 0^{th} character of first String < 0^{th} character of second String then return 0^{th} character of first String + recursive method call(rest of first String, second String) else return 0^{th} character of second String + recursive method call(first String, rest of second String)

4. Find longest repeated pattern
   Given a String, find the longest sequence of matching characters in that String and return that int length. For instance, "aaabbccccccdde" has 6 c's so the method would return 6
   Call this method passing it the first character in the String, the rest of the String and 1, as in findLongest(char c, String s, int length)
   length represents the longest matching sequence found so far, which is 1 because we found c
   Base case: if length==0 or length==1 return the length of s
   Recursive cases: if c matches the 0^{th} character of s, then we extend the sequence, return the value of the recursive call with c, the rest of s after the 0^{th} character, and length+1
   else no match, return whichever is larger, length (the length we found of char c up to this point in s), or the value returned by recursively calling the method with the 0^{th} character of s, the rest of s, and 1 as we are starting over
   Example: "aaabbccccccdde"
   Call the method with 'a', "aabbccccccdde", 1
   First character matches, so call with 'a', "abbccccccdde", 2
   First character matches, so call with 'a', "bbccccccdde, 3
   No match, so return max(3, call with 'b', "bccccccdde", 1)
   First character matches, so call with 'b', "ccccccdde, 2
   No match, so return max(2, call with 'c', "cccccdde", 1)
   First character matches, so call with 'c', "ccccdde", 2
   …
   Eventually these calls return 6 (6 c's found)
   Return max(2, 6)
   Return max(3, 6)

Run your program on the following. Hand in your (commented) source code and the result of running the program on these variations.

permutations: "abc", "12345"
alphabetic pairs: "aybcmznx", "lazybboy" (note: two b's in a row), "wisely"
merge: "aaccee" and "bcdef", "0159", "234678"
find longest sequence: "aabbbcddddddeefffghiii", "19855aa00000cc24443bbb"