CS102: Introduction to Computer Science
Summer 2013
Program #3

Two strings are anagrams of each other if they contain the same letters, in any order. (In other words, every letter of the output must occur the same number of times in both strings.) White space, punctuation, and other non-letter characters are ignored, and lowercase letters are treated as identical to uppercase letters. You are going to write an anagram detector which evaluates pairs of strings to determine if the two strings in each pair are anagrams of each other.

The program will read from standard input. The first input will be in integer representing the number of test cases to follow. Each of the following test cases will contain two strings, one string per line, and every string will end with a newline character. Every string is guaranteed to have less than 50 characters (so up to 50 characters including the added null character). For each test case (i.e., for each pair of strings), your program should display:

```
Test case #<number>
```

followed by a newline character. There should be no leading whitespace and no space after the '#' for such lines. The numbering should start at 1. Then, on two separate lines, the program should display:

```
   string1: <first string>
   string2: <second string>
```

with exactly three spaces to the left of "string1" and "string2", one space after each colon, and a single newline character at the end of each string. Then, on the next line, the program should display either:

```
   These strings are anagrams
```

or:

```
   These strings are NOT anagrams
```

with exactly three spaces to the left of either message. Of course, the first of the two messages should be displayed if and only if the two strings (string1 and string2) are anagrams.

You should write your program in C, of course. Use functions to make the program more readable and avoid repeated code. I suggest having one function to check if two passed in strings are anagrams of each other (returning either 1 if yes, 0 if no); that can rely on another function if both strings need to be processed in the same way (to avoid repeated code).

Hint: I think the simplest way to do this is to count how many times each letter of the alphabet occurs in each strings, and to store the 26 counts for each string in an array (use a different array for each of the two strings). You should only need to loop through the characters in each string once to do this. Then you just need to see if all of the counts match to determine if the two strings are anagrams (this can be checked with another simple loop). There are other possible strategies, of course, but you might lose points if you do something that is less efficient.

A sample run is shown on the next page. I will be redirecting standard input to come from a text file, and I will test on a data set larger than the sample run that is shown here. I will also redirect standard output to be written to a text file. If your output is not exactly the same as mine (compared using "diff"), you will lose points.

CS102: Introduction to Computer Science

Summer 2013

Program #3

Assuming that the input to the program is as follows:

```
5
dormitory
dirty room
dormitory
dirty, dirty room
William Shakespeare
I'll make a wise phrase
Hello World!
A common message
slot machines
*** Cash lost in 'em! ***
```

The output should look EXACTLY like this:

```
Test case #1
   string1: dormitory
   string2: dirty room
   These strings are anagrams
Test case #2
   string1: dormitory
   string2: dirty, dirty room
   These strings are NOT anagrams
Test case #3
   string1: William Shakespeare
   string2: I'll make a wise phrase
   These strings are anagrams
Test case #4
   string1: Hello World!
   string2: A common message
   These strings are NOT anagrams
Test case #5
   string1: slot machines
   string2: *** Cash lost in 'em! ***
   These strings are anagrams
```

Your homework will be graded out of 100 points with the following breakdown:

- **Correctness**: You should follow all instructions exactly as stated above. This is an individual assignment (i.e., you should not collaborate with anyone else). **75 points**.
- **Elegance and Efficiency**: You should use the concepts we have learned in class to write your program in a simple, elegant manner. Use functions to make the code more readable and to avoid repeated code. Avoid unnecessary processing. **10 points**.
- **Format**: Your program should use proper indentation and other spacing which makes the code readable and easy to understand. **10 points**.
- **Comments**: Include one comment at the top of the program explaining what the program does, and also your name. Include one comment above each function, explaining its purpose. Include one or more short comments within each function explaining the code. **5 points**.

Submitting assignments: Email me your code (to **CarlSable.Cooper@gmail.com**) as an attachment. Do NOT attach the executable (only send the source file). Please state in your e-mail which environment you used to develop the code (e.g., Cygwin, Quincy, Ubuntu, etc.). This program is due the night of Sunday, August 4, before midnight.