1. The most notable obstacle was for the separate function. Initially, I tried to make it so that if the value < separator, it would value would be placed in the front of the array and if value > separator, it would be placed in the end of the array (I realize now that by adding the second part, it makes coding a lot more difficult). This strategy made it hard to successfully implement the for loop, since the position of each element in the array kept changing. I fixed this problem by simply rearranging the elements in the array in alphabetical order by switching two neighboring elements at a time a total of n^2 times. I realize that this method is not very efficient, however.  
     
   Another obstacle was with subsequence. Although my code did work with most test cases, I realized that my code could potentially go out of bound if the first element of a2[] appears as the last element of a1[], and a2[] has more than one element. I fixed this issue by adding another condition in one of my if-statements to make sure that it never goes out of bound.

1. string test1[5] = {“Jan”, “Tom”, “Bob”, “Phil”, “Justin”};

string test2[0] = {};

string test3[10] = {“a”,”t”,”z”,”r”,”m”,”n”,”b”,”q”,”w”,”s”};

string test4[10] = {“Jack”, “Jill”};

string test5[10] = {“Jack”, “Jack”, “Jack”, “Bob”, “Trump”, “Trump”, “Hillary”, “Jill”, “Jill”,”Tom”};

string test6[10] = {“a”,”t”,”z”,”r”,”m”,”n”,”b”,”q”,”w”,”s”};

string test7[10] = {“a”,”t”,”z”,”r”,”l”,”n”,”b”,”q”,”w”,”s”};

string test8 [3] = {“Phil”, “Justin”, “Jack”};

string test9 [2] = {“Phil”, “Justin”};

**appendToAll**

string final1 = “!!!!!”;

string final2 = “”;

* appendToAll(test1, 5, final1) //Tests normal case
* appendToAll(test1, 3, final1) //Tests case where n < array length
* appendToAll(test2, 0, final1) //Tests a blank array
* appendToAll(test3, 8, final2) //Tests blank append
* appendToAll(test4, 4, final1) //Tests array with blank spots
* appendToAll(test1, -4,final1) //Tests a non-valid array length

**lookup**

string final1 = “Tom”;

string final2 = “r”;

string final3 = “negative”;

string final4 = “”;

* lookup(test1, 5, final1) //Tests normal case
* lookup(test3, 10, final3) //Tests case where there is no right value
* lookup(test4, -1, final2) //Tests case with non-valid array length
* lookup(test3, 2, final4) //Tests blank string

**positionOfMax**

* positionOfMax(test1, 5) //Tests normal case
* positionOfMax(test1, 3) //Tests case where n < array length
* positionOfMax(test, 0) //Tests blank string
* positionOfMax(test, -1) //Tests case with non-valid array length

**rotateLeft**

* rotateLeft(test1, 5, 3) //Tests normal case
* rotateLeft(test1, 3, 2) //Tests case where n < array length
* rotateLeft(test4, -1, 3) //Tests case with non-valid array length
* rotateLeft(test1, 0, 0) //Tests blank array

**countRuns**

* countRuns(test5, 10) //Tests normal case
* countRuns(test1, 5) //Tests case where there is no repeat
* countRuns(test5, 6) //Tests case where n < array length
* countRuns (test4, -1) //Tests case with non-valid array length
* countRuns (tests1, 0) //Tests blank array

**flip**

* flip(test1, 5) //Tests normal case
* flip(test1, 3) //Tests case where n < array length
* flip(test1, -3) //Tests case with non-valid array length
* flip(test1, 0) //Tests blank array

**differ**

* differ(test7, 10, test6, 10) //Tests normal case
* differ(test7, 3, test6, 10) //Tests case where n1 < n2
* differ(test7, 10, test6, 3) //Tests case where n2 < n1
* differ(test7, -1, test6, 10) //Tests case with non-valid array length
* differ(tests7, 10, test6, -1) //Tests case with non-valid array length
* differ(test7, 0, test6, 10) //Tests blank array
* differ(test7, 10, test6, 0) //Tests blank array
* differ(test3, 10, test6, 10) //Tests case where everything is the same

**subsequence**

* subsequence(test1, 5, test9, 2) //Tests normal case
* subsequence(test1, 3, test9, 2) //Tests case where n1 < a1[] length
* subsequence(test1, 5, test9, 1) //Tests case where n2 < a2[] length
* subsequence(test1, -1, test9, 1) //Tests case with non-valid array length
* subsequence(test1, 5, test9, -1) //Tests case with non-valid array length
* subsequence(test1, 0, test9, 2) //Tests blank array
* subsequence(test1, 5, test9, 0) //Tests blank array
* subsequence(test3, 10, test6, 10) //Tests case where everything is the same
* subsequence(test1, 5, test8, 3) //Tests case where it goes out of bounds

**lookupAny**

* lookupAny(test5, 10, test4, 2) //Tests normal case
* lookupAny(test1, 3, test3, 10) //Tests case where n1 < a1[] length
* lookupAny(test1, 5, test3, 5) //Tests case where n2 < a2[] length
* lookupAny(test1, -1, test3, 5) //Tests case with non-valid array length
* lookupAny(test1, 5, test3, -1) //Tests case with non-valid array length
* lookupAny(test1, 0, test3, 5) //Tests blank array
* lookupAny(test1, 5, test3, -1) //Tests blank array
* lookupAny(test1, 5, test3, 10) //Tests case where nothing is same

**separate**

* separate(test1, 5, “Jack”) //Tests normal case
* separate(test1, 5, “Bob”) //Tests case where separator is in array
* separate(test1, 3, “Jack”) //Tests case where n < array length
* separate(test1, -1, “Jack”) //Tests case with non-valid array length
* separate(test1, 0, “Jack”) //Tests blank array