1. A brief description of notable obstacles you overcame.
   1. I was struggling with the problem of ‘how to swap elements in an array without creating a new array’ to get the bonus points. Initially, I thought if I didn’t use any new array, I’m always going to lose one element since it will be overwritten by the other element that is being swapped. Then, I suddenly recalled that I could use an additional string variable, that temporarily stores the value at the 1st index, and move the value of the 2nd index to the 1st index, and finally put the value of the temporary string to the 2nd index. By doing this in a for loop, I can swap all the values in an array without creating a new array.
   2. When writing up the ‘flip’ function, I keep getting erroneous results when I’m testing my solution. After using the debugger to setup breakpoints and trace my code, I realized that I don’t have to loop throught the whole array – in fact, only half of it, since I am swapping two elements at the same time. So I quickly changed i<n; to i<n/2; in the for statement condition and fixed this error.
   3. I was stuck at the ‘divide’ function for a long time since I didn’t figure out an solution for the problem. Then, I thought of using a sorting algorithm to completely sort the array alphabetically – a solution to the problem. Although this sounds over-exhaustive solution to the problem which is definitely more than enough and has a time complexity of O(n^2), the result matches what was required by the project spec.
2. Test Cases
   1. reduplicate(string a[], int n)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." }; (Tests for empty characters, the ‘.’ Char, and regular strings.)
         1. int j = reduplicate(stuff, 6); (Test for iterating the whole array)
         2. int j = reduplicate(stuff, 3); (Test for iterating a part of the array)
         3. int j = reduplicate(stuff, 0); (Test for iterating none of whole array – no changes should be made)
         4. int j = reduplicate(stuff, -1); (Test for bad arguments, should return -1)
      2. string noStuff[0] = {}; (Tests for empty array)
         1. int j = reduplicate(noStuff, 0); (Test for iterating empty array – no changes should be made)
   2. locate(const string a[], int n, string target)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." };
         1. locate(stuff, 6, "CS") (Test for searching existing element, should return 0)
         2. locate(stuff, 3, "Test") (Test for searching a part of array, should return 2)
         3. locate(stuff, 3, ".") (Test for does not exist elements, should return -1)
         4. locate(stuff, -1, ".") (Test for bad arguments, should return -1)
   3. locationOfMax(const string a[], int n)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." };
         1. locationOfMax(stuff, 6); (Test for looping whole array, should return 2)
         2. locationOfMax(stuff, 0); (Test for case that examines no part of array – should return -1)
         3. locationOfMax(stuff, -1); (Test for bad arg – should return -1)
   4. circleLeft(string a[], int n, int pos)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." };
         1. circleLeft(stuff, 6, 1); (Test for looping through whole array, start with index 1)
         2. circleLeft(stuff, 3, 1); (Test for looping through part of array, start with index 1)
         3. circleLeft(stuff, 6, 5); (Test for looping through whole array, start with index 5 – it essentially makes no change to the array)
         4. circleLeft(stuff, 6, 6); (Index 6 does not exist – return -1)
   5. enumerateRuns(const string a[], int n)
      1. string specialStuff[8] = { "I", "I", "I", "love", "CS", "31", “!”, ”!”};
         1. enumerateRuns(specialStuff, 8); (test for looping the whole array – should return 5)
         2. enumerateRuns(specialStuff, 3); (test for looping the a part of array – should return 1)
         3. enumerateRuns(specialStuff, 0); (test for looping the none of the array – should return 0)
   6. flip(string a[], int n)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." };
         1. flip(stuff, 6); (test for flipping whole array)
         2. flip(stuff, 3); (test for flipping first 3 elements of array)
         3. flip(stuff, 1); (test for flipping the 1st elements of array - should not alter the array)
         4. flip(stuff, 0); (test for flipping the 1st elements of array – should not alter the array)
         5. flip(stuff, -1); (test for invalid arguments – should not run)
      2. string oneElement [1] = {"CS"};
         1. flip(stuff, 1); (test for flipping the 1st elements of array - should not alter the array)
         2. flip(stuff, 0); (test for flipping the 1st elements of array – should not alter the array)
   7. locateDifference(const string a1[], int n1, const string a2[], int n2)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." };  
         string differentStuff[5] = { "CS", "31", "", "Test", "Cases" };
         1. locateDifference(stuff, 6, differentStuff, 3); (Test for searching for full array) – should return 2.
         2. locateDifference(stuff, 1, differentStuff, 2); (Test for searching for part array) – should return 1. Array1 runs out.
         3. locateDifference(stuff, 0, differentStuff, 2); (Test for searching for part array) – should return 0. Array1 runs out.
         4. locateDifference(stuff, -1, differentStuff, -1); (Test for invalid input. Return -1)
   8. subsequence(const string a1[], int n1, const string a2[], int n2)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." };  
         string lessStuff[3] = { "31", "Test", ""};
         1. subsequence(stuff, 6, lessStuff, 3) (Test for looping through full arrays) – return 1
         2. subsequence(stuff, 6, lessStuff, 0) (Test for empty subsequence) – return 0
         3. subsequence(stuff, 6, lessStuff, -1) (Test for invalid input) – return -1
      2. string stuff[6] = { "31", "31", "Test", "", "Cases", "." };  
         string lessStuff[3] = { "31", "Test", ""};
         1. subsequence(stuff, 6, lessStuff, 3) (Test to check if program misrecognizes 31 at index 0 is the start of subsequence – should be index 1) – returns 1
   9. locateAny(const string a1[], int n1, const string a2[], int n2)
      1. string stuff[6] = { "CS", "31", "Test", "", "Cases", "." };  
         string lessStuff[2] = { "31", ""};
         1. locateAny(stuff, 6, lessStuff, 2); (Loop through whole array) – returns 1;
         2. locateAny(stuff, 0, lessStuff, 2); (Does not look at array1) – returns -1;
         3. locateAny(stuff, 6, lessStuff, 0); (Does not look at array2) – returns -1;
         4. locateAny(stuff, -1, lessStuff, 0); (Errornous input) – returns -1;
   10. divide(string a[], int n, string divider)
       1. string divideTest[6] = {"Bravo", "Zelda", "Alpha", "Y", "X", "Delta" };
          1. divide(divideTest,6,"A"); (Loop through the whole array – divider on the leftmost side) – return 0;
          2. divide(divideTest,6,"Zzzzz"); (Loop through the whole array – divider on the rightmost side) – return 5;
          3. divide(divideTest,5,"M"); (Loop through a part of array – divider on the mid side) – return 4;
          4. divide(divideTest,0,"M"); (does not loop through array) – return 0;
          5. divide(divideTest,-1,"M"); (invalid input ) – return -1;