1. Notable Obstacles
   1. Overall, most of the functions in this project were relatively straight forward in producing. The one that gave the most trouble, however, was the separate function. Originally, I used the rotateLeft function to push everything that was greater than the separator to the end of the array, but that allowed strings that were the same as the separator to get pushed to the very front. Thus, I allowed the rotateLeft function to also push those that equaled the separator. Additionally, I created a counter system that placed the string that equaled the separator where it’s supposed to go.
2. Test Code
   1. appendToAll
      1. string s[] = {“A”, “B”, “C”, “Hi”, “54”}
      2. (s, 5, “!!”)
         1. Checks to see if all 5 elements are appended properly
      3. (s, 3, “!!”)
         1. Checks to see if only 3 of the elements are appended properly
      4. (s, 0, “!!”)
         1. Used to see if program properly alters no elements
      5. (s, -1, “!!”)
         1. Used to see if the program returns -1 correctly
   2. lookup
      1. string a[] = {"K", "O", "U", "HI", "4"}
      2. (a, 3, “U”)
         1. Checks to see if program will find give the correct position for U
      3. (a, 3, “4”)
         1. Checks to make sure program returns -1 as 4 is out of user bounds
      4. (a, -2, “4”)
         1. Checks to see if program returns -1 correctly
   3. positionOfMax
      1. string persons[] = { "donald", "lindsey", "marie", "rudy", "fiona", "adam"}
      2. (persons, 3)
         1. Makes sure the array is limited by input bound of 3
      3. (persons, 6)
         1. Makes sure the array returns the proper location of the “largest” string
      4. (persons, -1) and (persons, 0)
         1. Makes sure the array returns -1
   4. rotateLeft
      1. string politician[5] = { "mike", "donald", "lindsey", "nancy", "adam" }
      2. (politician, 5, 1)
         1. Tests to see if the function works properly (check the array order after with a loop)
      3. (politician, 4, 1)
         1. Test to see if the function ignores the last element, “adam”, but properly moves the other elements
      4. (politician, 5, 2)
         1. Tests to see if the function properly handles element “lindsey” as well
      5. (politician, -1, 1)
         1. Tests to see if the function returns -1
   5. countRuns
      1. string d[9] = {"rudy", "adam", "mike", "mike", "fiona", "fiona", "fiona", "mike", "mike" }
      2. (d, 9)
         1. Tests to see if the function works properly with the entire look
      3. (d, 5)
         1. Tests to see if the function is properly bounded by user input
      4. (d, -5)
         1. Makes sure the array returns -1
   6. flip
      1. string folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" }
      2. (folks, 7)
         1. Tests to see if the function can flip the entire string properly
      3. (folks, 5)
         1. Tests to see if the function can flip a portion of the string properly
      4. (folks, -5)
         1. Tests to see if the function returns -1
   7. differ
      1. string folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" }

string group[6] = { "adam", "", "fiona", "donald", "mike", "rudy" }

* + 1. (folks, 7, group, 6)
       1. Checks to see if it can compare entire strings of differing sizes
    2. (folks, 2, group, 3) and (folks, 3, group 2)
       1. Checks to see if correct lower boundary is returned if string is identical up to them
    3. (folks, 0, group, 0) and (folks, -5, group 3) and (folks, 3, group -5)
       1. Checks to see if function returns -1 properly
  1. subsequence
     1. string names[10] = {"gordon", "marie", "nancy", "mick", "adam", "lindsey"}

string names1[10] = {"marie", "nancy", "mick"}

string names2[10] = { "gordon", "mick" };

* + 1. (names, 6, names1, 3)
       1. Checks to see if the function properly returns 1 (essentially testing if it works)
    2. (names, 5, names2, 2)
       1. Checks to see if the function returns -1 properly (no matches)
    3. (names, 0, names1, 0) && (names, 5, names1, 0)
       1. Checks to see if function returns 0
  1. lookupAny
     1. string names[10] = {"gordon", "marie", "nancy", "mick", "adam", "lindsey"}

string set1[10] = {"donald", "adam", "mick", "marie"}

string set2[10] = { "rudy", "fiona" };

* + 1. (names, 6, set1, 4)
       1. Checks to see if the function works and returns 1 (“marie”
    2. (names, 6, set2, 2)
       1. Checks to make sure the function returns -1 (no matches)
    3. (names, 0, set1, 2) and (names, 2, set1, 0)
       1. Checks to make sure it returns -1
  1. separate
     1. string persons[6] = { "donald", "lindsey", "marie", "rudy", "fiona", "adam" }

string persons2[4] = { "marie", "nancy", "lindsey", "mike" };

* + 1. (persons, 6, "gordon")
       1. Checks to see if the function works with a string that doesn’t exist in the array
    2. (persons2, 4, "mike") and (persons, 6, "marie")
       1. Checks to see if the function works with a string that already exists in the array
    3. (persons, 0, “gordon”) and (persons, -5, “gordon”)
       1. Checks to see if the function responds properly to 0 and negatives
    4. (persons, 4, “gordon)
       1. Checks to see if the function works with boundaries input by the user (4)