a. Obstacles:

* In the beginning, just familiarizing myself with arrays and what methods I would need to use to work through the elements of the array and then adapt it to fit each function
* The first notable obstacle I encountered was with the rotateLeft function, and creating an expression that would shift over all of the elements to the left
* With the flip function, I was able to use the temporary variable to swap values of elements in the array, but it was tricky to create a statement that would only iterate through the array for the first half
  + Originally I had k < n for the condition in the for loop, but this would only swap the values twice when I only wanted to swap the values once (for the first half of the array elements)
* With the subsequence function, it was tricky to compare the consecutive sequences of the second array with elements that could be in any starting position of the first array
  + Was able to do so by running through the elements of the second array and checking them with any position of the elements in the first array
* The last function, the split function, was definitely the most difficult function
  + Creating the markers and working through the array elements from the beginning and from the end was difficult to implement
  + By using pseudocode and realizing that there were three conditions that could occur, I was able to advance through the array elements and swap values to move items before or after the splitter
  + The previous function of flip also helped me to get the idea of what was supposed to be done

b. Test Data:

* For each function, if n < 0
  + To make sure that the array had negative elements
  + return -1
* For append function;
  + string g[4] = {"bernie", "hillary", "jeb", "carly"};
  + appendToAll (g, 4, "?")
    - To test when the appendage is a normal case
  + appendToAll (g, 4, "")
    - To test if nothing will be appended
  + appendToAll (g, 0, "")
    - To test if n is equal to zero

All tests succeeded

* For lookup function:
  + string h[7] = {"bernie", "hillary", "donald", "jeb", "", "carly", "ben"};
  + assert(lookup(h, 7, "carly") == 5);
    - To test looking up an element in a normal case
  + assert(lookup(h, 2, "donald") == -1);
    - To test looking up an element that is not present in the size being looked at
  + assert(lookup(h, 0, "donald");
    - To test if n is equal to 0

All tests succeeded

* For max position function:
  + string h[7] = {"bernie", "hillary", "donald", "jeb", "", "carly", "ben"};
  + assert(positionOfMax(h, 7) == 3);
    - To test what the max position would be in a normal case
  + assert(positionOfMax(h, 0) == -1);
    - To test what the max position would be if there are no array elements

All tests succeeded

* For rotate left function:
  + string g[4] = {"bernie", "hillary", "jeb", "carly"};
  + assert(rotateLeft(g, 4, 1) == 1 && g[1] == "jeb?" && g[3] == "hillary?);
    - To test if the elements will be rotated accordingly in a normal case
  + assert(rotateLeft(g, 1, 1) == 1);
    - To test if there is only 1 element in the array
  + assert(rotateLeft(g, 0, 1) == -1);
    - To test if there are no elements of interest

All tests succeeded

* For count runs function:
  + string e[5] = {"hillary", "hillary", "hillary", "ben", "ben"};
  + assert(countRuns(d, 5) == 2);
    - To test the number of runs in a normal case
  + string e[5] = {"hillary", "hillary", "hillary", "hillary", "hillary"};
  + assert(countRuns(d, 5) == 1);
    - To test the number of runs if there is only a repeated element in the array
  + assert(countRuns(d, 0) == 0);
    - To test if no elements of interest

All tests succeeded

* For flip function:
  + string f[3] = {"jeb", "donald", "marco"};
  + assert(flip(f, 3) == 3 && f[0] == "marco" && f[2] == "jeb");
    - To test what would happen if there is an odd number of elements
  + string e[4] = {"donald", "jeb", "", "carly"};
  + assert(flip(e, 4) == 4 && e[0] == "carly" && e[3] == "donald");
    - To test what would happen if there is an even number of elements
  + assert(flip(f, 0) == 0);
    - To test if there are no elements of interest

All tests succeeded

* For differ function:
  + string g[4] = {"bernie", "hillary", "jeb", "carly"};
  + assert(differ(h, 4, g, 4) == 2);
    - To test for a difference in a normal case
  + string f[3] = {"jeb", "donald", "marco"};
  + string g[4] = {"bernie", "hillary", "jeb", "carly"};
  + assert(differ(f, 1, g, 1) == 0);
    - To test if the first elements differ
  + assert(differ(f, 0, g, 0) == 0);
    - To test if there are no elements of interest

All tests succeeded

* For subsequence function:
  + string h[7] = {"bernie", "hillary", "donald", "jeb", "", "carly", "ben"};
  + string e[4] = {"donald", "jeb", "", "carly"};
  + assert(subsequence(h, 7, e, 4) == 2);
    - To test if there is consecutive sequence in a normal case
  + string f[3] = {"jeb", "donald", "marco"};
  + string g[4] = {"bernie", "hillary", "jeb", "carly"};
  + assert(subsequence(f, 3, g, 4) == -1);
    - To test if there is no consecutive sequence
  + string g[4] = {"bernie", "hillary", "jeb", "carly"};
  + string g[4] = {"bernie", "hillary", "jeb", "carly"};
  + assert(subsequence(g, 4, g, 4) == 0)
    - To test if both arrays are identical
  + assert(subsequence(g, 0, g, 0) == 0);
    - To test if there are no elements of interest

All tests succeeded

* For lookup any function:
  + string h[7] = {"bernie", "hillary", "donald", "jeb", "", "carly", "ben"};
  + string f[3] = {"jeb", "donald", "marco"};
  + assert(lookupAny(h, 7, f, 3) == 2);
    - To test to lookup any in a normal case
  + string h[7] = {"bernie", "hillary", "donald", "jeb", "", "carly", "ben"};
  + string f[3] = {"jeb", "donald", "marco"};
  + assert(lookupAny(h, 0, f, 3) == -1);
    - To test if there is an array testing for 0 elements

All tests succeeded

* For split function:
  + string h[7] = {"bernie", "hillary", "donald", "jeb", "", "carly", "ben"};
  + assert(split(h, 7, "carly") == 3);
    - To test for a correct split with one of the elements being equal to the splitter
  + string h[7] = {"bernie", "hillary", "donald", "jeb", "", "carly", "ben"};
  + assert(split(h, 7, "charlie") == 4);
    - To test for a correct split with no elements being equal to the splitter
  + assert(split(h, 0, "charlie") == 0);
    - To test if there are no elements of interest

All tests succeeded