**Notable obstacles during development**

1. While developing the *subsequence* function, I had to find a way to break out of the loop if the tested string in a1 was not equal to the corresponding string in a2. I first tried using a *break* statement as in Project 3, but I couldn’t figure out how to break out of multiple loops. A *goto* statement would have worked but is considered bad practice. Instead, I created a local Boolean, *hasSubsequence*, to keep track of whether the arrays should undergo further testing. If the corresponding elements are all equal, then *hasSubsequence* remains true and the function returns the position where the subsequence starts.
2. In the *flip* function, elements of the array would be swapped outside of the indicated boundaries. I realized that *n* counted the length of the array starting from 1, while the array index started counting at 0. I subtracted 1 from the mirrored position in the code, which rectified the issue.
3. For the *divide* function, I needed to find an algorithm to sort the array using the *divider* string as the pivot. This sounded similar to the pivot algorithm in Quicksort, which I tried to implement in the function. However, the input array contains strings instead of numbers and the number of elements is relatively small. As a result, I used bubble sort because it was robust enough for the task and easier to implement.

**Test data**

The program passes all of the test cases below.

appendToAll

string people[5] = {"dianne", "fiona", "ed", "xavier", "greg"};

* people, 5, !!!
  + n is equal to length of array
* people, -3, !!!
  + n is negative
* people, 0, !!!
  + n is 0

lookup

string s[4] = {“Bob”, “Jim”, “Sam”, “Ryan”};

* s, 4, “Jim”
  + target found in array
* s, 4, “sAM”
  + different capitalization from element in array
* s, 4, “Cole”
  + target not in array
* s, 4, “Ryan”
  + target at end of array
* s, 4, “Bob”
  + target at beginning of array
* s, -2, “Sam”
  + n is negative
* s, 0, “Jim”
  + n is 0

string t[5] = {“car”, “boat”, “car”, “car”, “flower”};

* t, 5, “car”
  + multiple instances of target in array

positionOfMax

string candidate[6] = {"dianne", "fiona", "gavin", "xavier", "ed", "betty"};

* candidate, -1
  + n is negative
* candidate, 0
  + n is 0
* candidate, 6
  + max occurs in middle of string

string colors[4] = {“violet”, “blue”, “green”, “brown”};

* colors, 4
  + max at beginning of string

string animals[3] = {“fish”, “cat”, “zebra”};

* animals, 3
  + max at end of string

string repeat[4] = {“abc”, “abc”, “abc”, “abc”}

* repeat, 4
  + all elements of array are identical

string caseMix = {“monitor”, “XYLOPHONE”, “REINDEER”, “keyboard”};

* mixture of uppercase and lowercase strings

rotateLeft

string politician[5] = {"eleni", "dianne", "fiona", "kevin", "gavin"};

* politician, 5, 1
  + pos between beginning and end of array
* politician, 5, 4
  + pos at end of array
* politician, -3, 4
  + n is negative
* politician, 0, 2
  + n is 0
* politician, 5, -1
  + pos is negative
* politician, 5, 0
  + pos is 0 (beginning of array)

countRuns

string d[9] = {"xavier", "betty", "john", "john", "ed", "ed", "ed", "john", "john"};

* d, 9
  + multiple sequences of consecutive identical items in array
* d, 3
  + items in array are unique
* d, 0
  + n is 0
* d, -2
  + n is negative

string water[4] = {“water”, “water”, “water”, “water”};

* water, 4
  + items in array are identical

flip

string folks[6] = { "betty", "john", "", "xavier", "kevin", "dianne" };

* folks, 6
  + n is equal to length of array
* folks, 4
  + n is even
* folks, 3
  + n is odd
* folks, 0
  + n is 0
* folks, -1
  + n is negative

differ

string folks[6] = {"betty", "john", "", "xavier", "kevin", "dianne"};

string group[5] = {"betty", "john", "dianne", "", "xavier"};

* folks, 6, group, 5
  + only some elements of a1 and a2 are equal
  + n1 is greater than n2
* folks, 3, group, 5
  + n2 is greater than n1
* folks, 2, group, 1
  + a1 and a2 are equal but different lengths
* folks, 2, group, 2
  + all elements of a1 and a2 are equal
  + a1 and a2 have same length
* folks, 3, group, 3
  + a1 and a2 are identical except for last element

string fruit[3] = {“apple”, “pear”, “strawberry”};

string vegetables[4] = {“lettuce”, “cauliflower”, “broccoli”, “spinach”};

* fruit, 3, vegetables, 4
  + no corresponding elements in a1 and a2
* fruit, -1, vegetables, 5
  + n1 is negative
* fruit, 2, vegetables, -5
  + n2 is negative

string names[5] = {“Josh”, “Zack”, “Brian”, “Neil”, “Jim”}

string moreNames[5] = {“Cameron”, “Zack”, “Brian”, “Neil”, “Sam”}

* names, 4, moreNames, 4
  + a1 and a2 are identical except for initial element
* names, 0, moreNames, 4
  + n1 is 0
* names, 4, moreNames, 0
  + n2 is 0

subsequence

string names[10] = {"eleni", "gavin", "kevin", "greg", "betty", "fiona"};

string names1[10] = {"gavin", "kevin", "greg"};

string names2[3] = {“betty”, “fiona”, “cole”};

* names, 6, names1, 3
  + a2 found in a1
* names, 4, names1, 3
  + a2 ends at last element of a1
* names, 0, names1, 3
  + n1 is 0
* names, 3, names1, 0
  + n2 is 0
* names, 0, names1, 0
  + both n1 and n2 are 0
* names, -1, names1, 3
  + n1 is negative
* names, 2, names1, -3
  + n2 is negative
* names, 6, names2, 3
  + a1 contains some of a2 but ends early

string s[5] = {“ab”, “cd”, “ef”, “gh”, “ij”};

string t[3] = {“kl”, “mn”, “op”};

* s, 3, t, 3
  + a1 does not contain a2 as a contiguous subsequence

lookupAny

string names[10] = {"eleni", "gavin", "kevin", "greg", "betty", "fiona"};

string set1[10] = {"dianne", "betty", "greg", "gavin"};

* names, 6, set1, 4
  + multiple elements of a2 in a1
* names, 6, set1, 1
  + no element of a1 is equal to any element of a2
* names, 0, set1, 4
  + n1 is 0
* names, 6, set1, 0
  + n2 is 0
  + sequence of 0 elements
* names, -1, set1, 4
  + n1 is negative
* names, 6, set1, -7
  + n2 is negative

divide

string candidate[6] = {"dianne", "fiona", "gavin", "xavier", "ed", "betty"};

* candidate, 6, “eleni”
  + value of divider is between elements of array
* candidate, 6, “zack”
  + value of divider Is greater than all elements of array
* candidate, 6, “andy”
  + value of divider is less than all elements of array
* candidate, 0, “fiona”
  + n is 0
* candidate, -2, “ed”
  + n is negative

string candidate1[4] = {“gavin”, “kevin”, “fiona”, “john”};

* candidate1, 4, “john”
  + divider equal to an element in array

swap

string infrastructure[4] = {“road”, “bridge”, “tunnel”, “rail”};

string vehicles[4] = {“car”, “train”, “boat”, “truck”};

* swap(infrastructure[1], vehicles[2])
  + elements in different arrays
* swap(vehicles[0], vehicles[3])
  + elements in same array