Notable Obstacles:

I had an obstacle implementing the divide function. My issue was handling when the divider string was in the array already. I had to implement an extra if case in my algorithm to ensure that the function properly sorted out all of the items.

Test Cases:

appendToAll:

string array[4] = {“hello”, “hello”, “hello”, “hello”};

**n is negative:**

assert (appendToAll(array, -1, “!!!”) == -1) – this test makes sure the function returns -1 when n < 0.

**n is valid:**

**n is array size:**

assert (appendToAll(array, 4, “!!!”) == 4 && array[0] == “hello!!!” && array[3] == “hello!!!”) –makes sure the function returns n when the n argument is valid and that the designated elements in the array have value added on to them.

**n is less than array size:**

assert (appendToAll(array, 2, “!!!”) == 2 && array[0] == “hello!!!” && array [2] == “hello”) – makes sure the function only appends value to the first n elements and not all of them.

lookup:

string a[3] = {“hi”, “bye”, “later”};

string b[3] = {“hi”, “hi”, “bye”};

**n is negative:**

assert (lookup(a, -1, “hi”) == -1) – makes sure the function returns -1 when n < 0.

**target appears once:**

assert (lookup(a, 3, “bye”) == 1) - makes sure the function works when the target string is present once in the array.

**target appears more than once:**

assert (lookup(b, 3, “hi”) == 0) - makes sure the function returns the position of the first appearance of target.

**target is not present:**

assert (lookup(b, 3, “yes”) == -1) - makes sure the function returns -1 when target is not in the array.

positionOfMax:

string a[4] = {“alpha”, “beta”, “zebra”, “grumpkin”};

string b[1] = {“”};

**n <= 0:**

assert (positionOfMax(a, 0) == -1) - makes sure function returns -1 when n is recognizably invalid.

**n is valid:**

assert (positionOfMax(a, 1) == 2) - makes sure function returns the proper position when n is valid.

rotateLeft:

string a[4] = {“apple”, “banana”, “orange”, “pear”};

**n <= 0:**

assert (rotateLeft(a, 0, 0) == -1) - makes sure function returns -1 when n is recognizably invalid.

**pos < 0:**

assert (rotateLeft(a, 4, -1) == -1) - makes sure function returns -1 when pos < 0.

**pos >= n:**

assert (rotateLeft(a, 3, 3) == -1)

assert (rotateLeft(a, 3, 4) == -1) - makes sure function returns -1 when pos >= n.

**valid arguments:**

assert (rotateLeft(a , 4, 1) == 1 && a[1] == “orange” && a[3] == “banana”) - makes sure function returns pos and correctly manipulates the array.

countRuns:

string a[6] = {“betty”, “ed”, “ed”, “ed”, “john”, “john”};

**n < 0:**

assert (countRuns(a, -1) == -1) - makes sure function returns -1 when n is recognizably invalid.

**n is valid:**

assert (countRuns(a, 6) == 3) - makes sure function returns proper number of runs when n is valid.

flip:

string a[5] = {“betty”, “john”, “dianne”, “bob”, “ross”};

**n < 0:**

assert (flip(a, -1) == -1) - makes sure function returns -1 when n is recognizably invalid.

**n == 0:**

assert (flip(a, 0) == 0 && a[0] == “betty” && a[4] == “ross”) - makes sure function doesn’t manipulate the array if n == 0.

**even n:**

assert (flip(a, 4) == 4 && a[0] == “bob” && a[3] == “betty” && a[4] == “ross”) - makes sure function works when n is even.

**odd n:**

assert (flip(a, 5) == 5 && a[0] == “ross” && a[4] == “betty”) - makes sure function works when n is odd.

differ:

string a[6] = {“dennis”, “fred”, “henry”, “greg”, “sam”, “adrian”};

string b[6] = {“dennis”, “fred”, “henry”, “greg”, “drake”, “anthony”};

string c[3] = {“dennis”, “fred”, “henry”};

string d[7] = {“dennis”, “fred”, “henry”, “greg”, “sam”, “adrian”, “adam”};

**n1 < 0:**

assert (differ(a, -1, b, 2) == -1) - makes sure function works when n1 < 0.

**n2 < 0:**

assert (differ(a, 6, b, -1) == -1) - makes sure function works when n2 < 0.

**n1 && n2 < 0:**

assert (differ(a, -1, b, -1) == -1) - makes sure function works when both n1 and n2 < 0.

**different arrays:**

assert (differ(a, 6, b, 6) == 4) - makes sure function works properly to return the position of first different value between the two arrays.

**no difference n1 > n2, no difference n2 > n1:**

assert (differ(a, 6, c, 3) == 3)

assert (differ(a, 6, d, 7) == 6) - makes sure function returns the lower of the two n’s when there is no difference between the arrays.

subsequence:

string a[6] = {“dennis”, “fred”, “henry”, “greg”, “sam”, “adrian”};

string b[6] = {“fred”, “henry”, “greg”};

string c[8] = {“a”, “b”, “c”, “d”, “e”, “f”, “g”, “h”};

string d[6] = {“a” “a”, “b”, “b”, “a”, “a”};

string e[2] = {“a”, “a”};

**n1 < 0, n2 < 0, n1 && n2 < 0 :**

assert (subsequence(a, -1, b, 3) == -1)

assert (subsequence(a, 6, b, -1) == -1)

assert (subsequence(a, -1, b, -1) == -1) - makes sure function returns -1 when n1 and/or n2 is negative.

**n1 == 0:**

assert (subsequence(a, 0, b, 3) == -1) - makes sure function returns -1 when n1 is 0.

**n2 == 0, n1 && n2 == 0:**

assert (subsequence(a, 6, b, 0) == 0)

assert (subsequence(a, 0, b, 0) == 0) - makes sure function returns 0 when n2 == 0.

**no sequence found && n1 > n2, no sequence found && n2 > n1:**

assert (subsequence(a, 6, c, 4) == -1)

assert (subsequence(a, 6, c, 8) == -1) - makes sure function returns -1 when a2 is not present in a1.

**sequence found, sequence found more than once:**

assert (subsequence(a, 6, b, 3) == 1)

assert (subsequence(d, 6, e, 2) == 0) - makes sure function returns earliest starting position of a2 sequence within a1.

lookupAny:

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

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

string c[10] = { "xavier", "ed" };

**n1 < 0, n2 < 0, n1 && n2 < 0:**

assert (lookupAny(a, -1, b, 4) == -1)

assert (lookupAny(a, 6, b, -1) == -1)

assert (lookupAny(a, -1, b, -1) == -1) - makes sure function returns -1 when n1 and/or n2 is negative.

**no equal element:**

assert (lookupAny(a, 6, c, 2) == -1) - makes sure function returns -1 when there is no equal element in the arrays.

**equal element found:**

assert (lookupAny(a, 6, b, 4) == -1) - makes sure function returns the pos in a1 where the equal element is found.

divide:

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

string candidate2[4] = { "gavin", "kevin", "fiona", "john" };

string candidate3[5] = { “john", ”gavin", "kevin", "fiona", "john" };

string a[2] = {“a”, “a”};

string b[2] = {“z”, “z”};

**n < 0:**

assert (divide(candidate, -1, “eleni”) == -1)

**n == 0:**

assert (divide(candidate, 0, “eleni”) == 0)

**n is valid:**

assert (divide(candidate, 6, “eleni”) == 3)

**divider is in the array:**

assert (divide(candidate2, 4, “john”) == 2)

**divider is in the array more than once:**

assert (divide(candidate3, 5, “johni”) == 2)

**all elements < divider:**

assert (divide(a, 2, “eleni”) == 2)

**all elements >= divider:**

assert (divide(b, 2, “eleni”) == 0)