a.

An obstacle that I had to overcome would be trying to implement the subsequence function. I was trying to think about how to approach it and then I realized that if I made a variable that was like a counter, it would be able to check if the sequence from a2 was fully completed in array a1 by seeing if the counter equaled n2. Another obstacle would be figuring out how to not change all the variables up to int n in the flip function. I realized that we are only switching up to n/2 rather than n because if it went up to n, it would redo the flip back to the original array. I implemented i< n/2 rather than i< n in the for loop. The last major obstacle that I had to face was trying to figure out how to make the function stay inbound of the array’s limit in the RotateLeft function. I decided to check if one index after the position existed prior to switching the current position of the index with one that was to the right of it.

b.

string h[7] = { "tiffany", "ryan","grace","sacha","gabe", "tyler", "simona" };

assert(lookup(h, 7, "tiffany") == 0); //checks if it'll return the correct position

assert(lookup(h, 7, "ryan") == 1); //checks if it'll return the correct position

assert(lookup(h, 7, "tyler") == 5); //checks if it'll return the correct position

assert(lookup(h, -1, "tiffany") == -1); //checks if it'll return -1 because of the negative n value

assert(lookup(h, 3, "tyler") == -1); //checks if it'll return -1 since tyler isn't in the first 3 elements

assert(positionOfMax(h,7) == 5); //checks if the max is correctly executed (tyler > tiffany)

assert(positionOfMax(h, -10) == -1); //checks if it'll return -1 because of a bad argument

assert(positionOfMax(h, 2) == 0); //checks if it can only take in an integer that equals the amount of elements in the array (are they checking when n is less than the amount of elements)

assert(appendToAll(h, 7, "!!!") == 7 && h[0] == "tiffany!!!" && h[2] == "grace!!!"); //checks if it adds the string correctly

assert(appendToAll(h, 7, "!") == 7 && h[0] == "tiffany!!!!" && h[2] == "grace!!!!"); //checks if it'll add a string on top of the modified array

assert(countRuns(h, 7) == 7); //checks if each element is counted as one item

string fruits[4] = { "peach", "cherry","strawberry","mango"};

assert(countRuns(fruits, 4) == 4); //checks that each index is one consecutive item

assert(countRuns(fruits, 2) == 2); //checks that each index is one consecutive item up to 2 elements

assert(flip(fruits, 4) == 4 && fruits[0] == "mango" && fruits[2] == "cherry"); //checks if the entire array flips

assert(flip(fruits, -4) == -1); //negative n value flip

//assert(flip(fruits, 2) == 2 && fruits[0] == "cherry" && fruits[2] == "strawberry"); //checks if part of an array can be flipped without messing up the other part of the array

assert(differ(h, 4, fruits, 4) == 0); //checks that it differs at the 0 index

assert(rotateLeft(fruits, -3, 1) == -1); //checks if it'll return -1 because of the negative n value

assert(rotateLeft(fruits, 3, -1) == -1); //checks if it'll return -1 because of the negative pos value

assert(lookupAny(h, 7, fruits, 4) == -1); //checks if there's no elements in fruits that are the same in h, return -1

string a[7] = { "rishi", "margaret", "gordon", "tony", "", "john", "liz" };

assert(lookup(a, 7, "john") == 5); //checks if it'll return the correct position

assert(lookup(a, 7, "gordon") == 2); //checks if it'll return the correct position

assert(lookup(a, 2, "gordon") == -1); //checks if it'll return -1 since gordon isn't in the first 2 elements

assert(positionOfMax(a, 7) == 3); //checks that tony is the max of the string

string g[4] = { "rishi", "margaret", "liz", "theresa" };

string g2[6] = { "rishi", "margaret", "liz", "theresa", "extra","extra"};

assert(differ(a, 4, g, 4) == 2); //shows that the string differs at index 2

assert(differ(g, 4, g2, 6) == 4); //checks that differ will return the lower value of n1 and n2 when strings run out and it's not different

//assert(differ(a, -4, g, -4) == -1); //checks if it'll return -1 because of the negative n value

assert(appendToAll(g, 4, "?") == 4 && g[0] == "rishi?" && g[3] == "theresa?");//adds ? to each string of the index

assert(rotateLeft(g, 4, 1) == 1 && g[1] == "liz?" && g[3] == "margaret?");//checks that it correctly rotates

string e[4] = { "gordon", "tony", "", "john" };

assert(subsequence(a, 7, e, 4) == 2); //checks that the subsequence does exist and it begins at index 2

assert(subsequence(h, 7, e, 4) == -1); //checks that if the subsequence doesn't exist in the main sequence, -1 will be returned

string d[5] = { "margaret", "margaret", "margaret", "tony", "tony" };

assert(countRuns(d, 5) == 2); // checks that there are 2 continuous segments of items

//assert(positionOfMax(d,5) == 4); //checks that it returns the first largest position

string f[3] = { "liz", "gordon", "tony" };

//assert(lookupAny(a, 7, f, 3) == 2); //checks the first index in a that has an item in f

//assert(flip(f, 3) == 3 && f[0] == "tony" && f[2] == "liz"); //checks if it'll flip an array with an odd amount of integers correctly

assert(split(f, 3, "liz") == 2); //checks if split works

string same[7] = { "help","help","help","help","help","help","help" };

assert(split(same, 7, "help") == 7); //checks if split will return n if nothing is greater than splitter

assert(split(same, -1, "help") == -1); //checks if -1 will return because of negative n value