### ***Notable Obstacles I Overcame***

1. For the *subsequent* function, I struggled with the logic for when to stop looping and checking if a2 was in a1. My first approach used an unnecessary amount of if statements in an attempt to check each element after a1[i], but soon realized that the array could have more than the small amount of elements my code covered. Each if statement could handle one element in an array. Finally, I realized I could take where my loops first identified a match a1[i], so position *i*, and add that to my inner loop iterator *k* in order to loop through both loops sequentially, and then return a value thus ending my functions processes. Next time I come across a problem that asks me to iterate when meeting a condition, then iterate through multiple arrays at the same time, I will remember this struggle.
2. For the split function, I recalled the function I wrote for the zybook participation assignment. We had to move elements in an array to the left and append the first element in the array to the end. Similarly, I hoped to adapt this function to move strings left if less than *splitter* and move them right if greater than. I struggled to do so for a while, until I changed my approach. My solution is to sort the strings alphabetically first, done with a simpler algorithm, then as soon as the *splitter* is less than the checked string value, then return the position *i* right before the *splitter*. When working on projects in the future, I will think about this and realize sometimes it isn’t my code that needs to change, it is the way I thought about solving the problem or task.

### ***Test Cases -* Reason for each test is highlighted and commented**

***\*Functions handled all these test cases correctly***

Int main()

{

//Given test cases

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

assert(lookup(h, 7, "john") == 5);

assert(lookup(h, 7, "gordon") == 2);

assert(lookup(h, 2, "gordon") == -1);

assert(positionOfMax(h, 7) == 3);

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

assert(differ(h, 4, g, 4) == 2);

assert(appendToAll(g, 4, "?") == 4 && g[0] == "rishi?" && g[3] == "theresa?");

assert(rotateLeft(g, 4, 1) == 1 && g[1] == "liz?" && g[3] == "margaret?");

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

assert(subsequence(h, 7, e, 4) == 2);

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

assert(countRuns(d, 5) == 2);

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

assert(lookupAny(h, 7, f, 3) == 2);

assert(flip(f, 3) == 3 && f[0] == "tony" && f[2] == "liz");

assert(split(h, 7, "liz") == 3);

cerr << "Passed Given Cases" << endl;

//Check Function 1

string a[5] = { "charlie", "gary", "riley", "lizelle", "jared" };

assert(appendToAll(a, -2, "pm") == -1);

//return -1 because -2 is negative

cerr << "YAY FUNCTION 1" << endl;

//no further testing as given function covers cases I would cover

//Check Function 2

const string a2a[5] = { "charlie", "gary", "riley", "lizelle", "jared" };

assert(lookup(a2a, -1, "lizelle") == -1);

//return -1 because -3 is negative

assert(lookup(a2a, 0, "charlie") == -1);

//return -1 because 0 implies no elements to look at

assert(lookup(a2a, 4, "lol") == -1);

//return -1 because “lol” is not in the array

assert(lookup(a2a, 2, "lizelle") == -1);

//return -1 because “lizelle” is not present in the first 2 elements

assert(lookup(a2a, 5, "jared") == 4);

//return 4 because “riley” is present at index 4

cerr << "YAY FUNCTION 2" << endl;

//Check Function 3

const string a3[5] = { "charlie", "gary", "riley", "lizelle", "jared" };

assert(positionOfMax(a3, -3) == -1);

//return -1 because -2 is negative

assert(positionOfMax(a3, 2) == 1);

//return 2 because gary is at index 1

assert(positionOfMax(a3, 5) == 2);

//return 2 because riley is at index 2

assert(positionOfMax(a3, 1) == 0);

const string a4[5] = { "", "oobleck", "tiana", "percy", "kevin" };

assert(positionOfMax(a4, 5) == 2);

//empty string

const string a5[5] = { "leopard", "leopard", "leopard", "leopard", "leopard" };

assert(positionOfMax(a5, 5) == 0);

//returns 0 because all indexes contain the same value

assert(positionOfMax(a5, 0) == -1);

//return -1 because function has no interesting elements

cerr << "YAY FUNCTION 3" << endl;

//Check Function 4

string a6[5] = { "charlie", "gary", "riley", "lizelle", "jared" };

assert(rotateLeft(a6, -1980, 2) == -1);

//impossible to rotate when n is negative

assert(rotateLeft(a6, 5, 5) == -1);

// cannot rotate out of index

assert(rotateLeft(a6, 5, -1980) == -1);

//index cannot be negative

cerr << "YAY FUNCTION 4" << endl;

//Check Function 5

string a7[6] = { "tony", "charlie", "riley", "riley", "lizelle", "lizelle" };

assert(countRuns(a7, -2) == -1);

//-2 is negative so should return -1

assert(countRuns(a7, 5) == 4);

//check for 3 same elements

string a8[6] = { "lol", "lol", "lol" };

assert(countRuns(a8, 0) == 0);

//array of 0 elements has 0 seq

assert(countRuns(a8, 3) == 1);

//should return 1 (only 1 unique element)

cerr << "YAY FUNCTION 5" << endl;

//Check Function 6

string a9[5] = { "charlie", "gary", "riley", "lizelle", "jared" };

assert(flip(a9, -2) == -1);

//return -1 because -2 is negative

assert(flip(a9, 0) == 0);

//return 0 because technically an empty array is valid

cerr << "YAY FUNCTION 6" << endl;

//Check Function 7

const string a10[7] = { "oobleck", "jared", "lizellez", "tiffany", "oobleck", "seinfeld", "tiana" };

const string a11[4] = { "oobleck", "jared", "lizellez", "kendall" };

assert(differ(a10, -2, a11, 5) == -1);

//return -1 because -2 is negative

assert(differ(a10, 7, a11, -1) == -1);

//return -1 because -1 is negative

const string a12[3] = { "lol", "lol", "lmao" };

const string a13[4] = { "lol", "lol", "lmao", "lmao" };

assert(differ(a12, 3, a13, 4) == 3);

//return 3 because 3<4

const string a14[3] = { "I", "am", "him" };

const string a15[1] = { "am" };

assert(differ(a14, 3, a15, 1) == 0);

//returns 0 because a14 doesn't start with the

cerr << "YAY FUNCTION 7" << endl;

//Check Function 8

string a16[6] = { "jared", "margaret", "theresa", "riley", "charlie", "lizelle" };

string a17[3] = { "margaret", "theresa", "riley" };

assert(subsequence(a16, 6, a17, 3) == 1);

//returns 1 because it starts at position 1

string a18[2] = { "hay", "styles" };

assert(subsequence(a16, 5, a18, 2) == -1);

//returns -1 because such a subsequence is not present

assert(subsequence(a16, 0, a17, 0) == 0);

//a sequence of 0 elements to be a subsequence of any sequence, even one with no elements, //starting at position 0

assert(subsequence(a16, 5, a17, 0) == 0);

//a sequence of 0 elements to be a subsequence of any sequence, even one with no elements, //starting at position 0

assert(subsequence(a16, 0, a17, 1) == -1);

//invalid entry because n1>n2

cerr << "YAY FUNCTION 8" << endl;

//Check Function 9

string a19[6] = { "jared", "theresa", "david", "charlie", "lizelle", "riley" };

string a20[3] = { "david", "charlie", "lizelle" };

assert(lookupAny(a19, 6, a20, 1) == 2);

//returns 2 as david is in position 2

string a21[3] = { "oobleck", "tiana", "biden" };

assert(lookupAny(a19, 6, a21, 3) == -1);

//return -1 as none found

assert(lookupAny(a19, 0, a20, 3) == -1);

//return 0 because an empty array will not have elements in any other array

assert(lookupAny(a19, 0, a20, 0) == -1);

//return 0 because an empty array will not have elements in any other array

cerr << "YAY FUNCTION 9" << endl;

//Check Function 10

string a22[6] = { "jared", "theresa", "david", "charlie", "lizelle", "riley" };

assert(split(a22, 6, "jared") == 2);

//return 2 because "jared" is the first one which is greater than or equal to "jared" in the //re-ordered array

string a23[6] = { "a", "b", "c", "d", "e" };

assert(split(a23, 4, "f") == 4);

//no elements less than f, should return 4

assert(split(a23, 0, "LA") == 0);

//empty array is valid, but because no element is >= to LA, should return 0

string a24[6] = { "jared", "theresa", "david", "charlie", "lizelle", "riley" };

assert(split(a24, 6, "korea") == 3);

//"lizelle" is the first one that is >=korea in the re-ordered array

cerr << "YAY FUNCTION 10" << endl;

cerr << "All tests succeeded" << endl;

}