a. Notable obstacles include figuring out why my code failed to run when I used the sample assert test main code. The debug would tell me that there was an error at line 29 which revolved around the appendToAll function. I thought this one was correct so I reread my code and tested it with different values and it seemed to be working correctly so I was stumped. I then made a test case where I would return the value and it returned 0. I looked back at the code and it was looking for a value of 4. Then I realized that I had my code return 0 if everything was good instead of returning n which I should have done originally. After changing this line of code, the assert cases were okay, and I was successful. Also, the separate function had me a bit stumped. I came to a problem during my logic where the function would not be able to figure out whether or not it was done since I had the function call the other function rotateLeft if the string at the current position was greater than the separator variable. This would result in an infinite call to rotateLeft and cause a major problem. I solved this problem by creating a boolean and another for loop within my for loop which would check for whether or not the rest of the strings in the array are greater than the variable separator. If they were, I would change the boolean to true which would then break out of the for loop and cause the function to be finished. This took me a lot of thinking in order to come up with, and I am extremely proud of my endeavor.

b. Test Cases

int appendToAll(string a[], int n, string value);

* string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int j = appendToAll(cast, 6, "!!!"); // adding excitement

int k = appendToAll(cast, 6, "-yessum"); // adding accent

int l = appendToAll(cast, -1, "asdf"); // negative array size

int m = appendToAll(cast, 3, " the Conquerer"); //only adding to the first 3 strings

int n = appendToAll(cast, 0, "quack"); // append to nothing

int lookup(const string a[], int n, string target);

* string look[5] = { "trex", "youngseo", "korea", "phantoml0rd", "rumble" };

int a = lookup(look, 5, "rumble"); // looks for "rumble"

int b = lookup(look, 3, "rumble"); // looks for something beyond scope (return -1)

int c = lookup(look, 5, "rUmBlE"); // has uppercase and not equal to "rumble"

int d = lookup(look, -3, "korea"); // negative array size, return -1

int e = lookup(look, 5, "DESTRUCTION"); // DESTRUCTION not found

int f = lookup(look, 0, "asdf"); // need somewhere to look up

int g = lookup(look, 5, "Youngseo"); // capitalization leads to no result

int h = lookup(look, 5, "t-rex"); // inclusion of hyphen leads to no result

int positionOfMax(const string a[], int n);

* string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int a = positionOfMax(cast, 6); // return 3, rick is latest in alphabetic order

int b = positionOfMax(cast, -100); // negative array size, return -1

int c = positionOfMax(cast, 3); // return 0, only check first 3 strings

int d = positionOfMax(cast, 0); // array has no elements, return -1

int rotateLeft(string a[], int n, int pos);

* string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int a = rotateLeft(characters, 5, 1); // shifts bob to end of array, return 1

int b = rotateLeft(characters, -5, 1); // negative array size, return -1

int c = rotateLeft(characters, 3, 1); // looks at first 3 strings of the array to perform action

int d = rotateLeft(characters, 0, 2); // rotate array with no elements, return -1

int rotateRight(string a[], int n, int pos);

* string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int a = rotateRight(characters, 5, 2); // return 2, move sasha to beginning and shift array values to the right

int b = rotateRight(characters, -15, 3); // negative array size, return -1

int c = rotateRight(characters, 0, 3); // rotate array with no elements, return -1

int d = rotateRight(characters, 5, 0); // basically does nothing, but legal

int flip(string a[], int n);

* string roles[6] = {"abraham", "tara", "", "daryl", "carol", "tyreese"};

int a = flip(roles, 4); // return 4, flips from abraham to daryl

int b = flip(roles, -5); // negative array size, return -1

int c = flip(roles, 6); // reverses the entire array, return 6

int d = flip(roles, 0); // reverse nothing, does nothing, return 0

int differ(const string a1[], int n1, const string a2[], int n2);

* string roles[6] = {"abraham", "tara", "", "daryl", "carol", "tyreese"};

string group[5] = { "abraham", "tara", "tyreese", "", "maggie" };

int r = differ(roles, 6, group, 5); // return 2, check both arrays with max int

int s = differ(roles, 2, group, 1); // return 1, check portions of both arrays

int t = differ(roles, -5, group, 5); // return -1, check first array with negative size

int u = differ(roles, 3, group, -3); // return -1, check second array with negative size

int v = differ(roles, -1, group, -2); //return -1, check both arrays with negative size

int w = differ(roles, 2, group, 4); // first array ends earlier than second array

int x = differ(roles, 3, group, 0); // checks for no possible string

int subsequence(const string a1[], int n1, const string a2[], int n2);

* string names[10] = { "sasha", "rick", "beth", "glenn", "bob", "michonne" };

string names1[10] = { "rick", "beth", "glenn" };

int t = subsequence(names, 6, names1, 3); // returns 1, subsequence found

string names2[10] = { "sasha", "glenn" };

int u = subsequence(names, 5, names2, 2); // returns -1, subsequence not found

int v = subsequence(names, -5, names2, 3); // returns -1, negative first array size

int w = subsequence(names, 3, names2, -3); // returns -1, negative second array size

int x = subsequence(names, -1, names2, -1); // returns -1, both arrays have negative size

int y = subsequence(names, 0, names2, 0); // both array have no elements

int z = subsequence(names, 5, names2, 0); // subsequence of no elements

int lookupAny(const string a1[], int n1, const string a2[], int n2);

* string names[10] = { "sasha", "rick", "beth", "glenn", "bob", "michonne" };

string set1[10] = { "maggie", "bob", "glenn", "rick" };

int v = lookupAny(names, 6, set1, 4); // returns 1 (a1 has "rick" there)

string set2[10] = { "daryl", "carol" };

int w = lookupAny(names, 6, set2, 2); // returns -1 (a1 has none)

int x = lookupAny(names, -1, set2, -1); // negative array sizes, return -1

int y = lookupAny(names, 6, set1, 1); // not in scope, cannot find maggie, return -1

int z = lookupAny(names, 0, set1, 0); // look to see if nothing is in nothing (no)

int q = lookupAny(names, 3, set1, 0); // check for nothing in something (no)

int separate(string a[], int n, string separator);

* string cast[6] = { "maggie", "carl", "daryl", "rick", "michonne", "carol" };

int x = separate(cast, 6, "glenn"); // returns 3

string cast2[4] = { "carol", "rick", "michonne", "daryl" };

int y = separate(cast2, 4, "daryl"); // returns 1

int z = separate(cast2, 3, "chris"); // doesn't include "daryl" in process

int s = separate(cast2, -3, "hello"); // negative array size, return -1

int t = separate(cast2, 0, "apu"); // separate looking at nothing