1. The most difficult part I found about writing the library was interpreting what happened for each function if it was given 0 important elements. I went through each function multiple times to make sure my interpretation made sense. Some of them were obvious, but I feel like a few were less so. I also had to go through a few times and tweak my comparison signs to make sure various for loops cut off when they were supposed to and if statements were triggered starting at the point they needed to. I had to make sure to read the spec carefully to distinguish when it was talking about position and when it was talking about the number of strings in the array to make sure my for loops didn’t go one extra time if I accidently made it go through until a[number of strings in array].
2. Test cases with solutions (organized by function execution)

appendToAll

string a[8] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina" };

assert(appendToAll(a, 8, "?") == 8 && a[0] == "selina?" && a[3] == "tony?" && a[4] == "?" && a[7] == "selina?"); //make sure it appends all the way to the end

assert(appendToAll(a, 2, "!!") == 2 && a[0] == "selina?!!" && a[1] == "reed?!!" && a[2] == "diana?" && a[3] == "tony?" && a[4] == "?" && a[7] == "selina?"); //make sure it only appends up to the point specified w/o confusion between "position" and n

assert(appendToAll(a, 0, "!!") == 0 && a[0] == "selina?!!" && a[1] == "reed?!!" && a[2] == "diana?" && a[3] == "tony?" && a[4] == "?" && a[7] == "selina?"); //make sure it changes nothing and returns 0 when given n = 0

lookup

string b[8] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina" };

assert(lookup(b, 8, "logan") == 5); //basic test

assert(lookup(b, 3, "logan") == -1); //test when array contains the string, but we don't check that far

assert(lookup(b, 8, "selina") == 0); //test first position

assert(lookup(b, 8, "august") == -1); //test something that isn't in the array

assert(lookup(b, 8, "") == 4); //check empty string

assert(lookup(b, 0, "logan") == -1); //test what happens if zero places are checked even though the array contains the string

assert(lookup(b, 0, "zzzzz") == -1); // same as above, but when the array does not contain the string

assert(lookup(b, 8, "Logan") == -1); //make sure it differentiates capitalization

assert(lookup(b, 1, "selina") == 0); //check only 1 place

positionOfMax

string c[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(positionOfMax(c, 9) == 3); //basic test of all places

assert(positionOfMax(c, 4) == 3); //check only some places

assert(positionOfMax(c, 0) == -1); //check no elements returns -1 as per the spec

assert(positionOfMax(c, 2) == 0); //check only up to a certain point

assert(positionOfMax(c, 1) == 0); //check only up to a certain point

string ca[8] = { "selina", "reed", "diana", "", "logan", "peter", "selina", "august" };

assert(positionOfMax(ca, 8) == 0); //make sure only the first occurance of the max is returned

rotateLeft

string d[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(rotateLeft(d, 4, 1) == 1 && d[1] == "diana" && d[2] == "tony" && d[3] == "reed" && d[8] == "august"); //check when only part is rotated

string da[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(rotateLeft(da, 0, 5) == -1); //rotating nothing returns -1

assert(rotateLeft(da, 9, 0) == 0 && da[0] == "reed" && da[8] == "selina"); // basic test with pos being the first string

countRuns

string e[8] = { "tim", "tim", "tim", "tim", "tim", "tim", "tim", "tim" };

assert(countRuns(e, 8) == 1); //all strings are the same

string ea[8] = { "tim", "august", "stacy", "stacy", "stacy", "amanda", "amanda", "august" };

assert(countRuns(ea, 8) == 5); //test reccurring strings

string eb[8] = { "tim", "August", "stacy", "Stacy", "stacy", "amanda", "amaNda", "august" };

assert(countRuns(eb, 8) == 8); //test effects of capitalization

assert(countRuns(eb, 0) == 0); //zero runs in a blank array

flip

string f[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(flip(f, 9) == 9 && f[0] == "august" && f[4] == "" && f[8] == "selina"); //basic test

string fa[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(flip(fa, 3) == 3 && fa[0] == "diana" && fa[1] == "reed" && fa[2] == "selina" && fa[3] == "tony"); //test when only some parts are flipped

string fb[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(flip(fb, 0) == 0 && fb[0] == "selina" && fb[8] == "august"); //test when nothing is flipped

differ

string g[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

string gx[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "tim" };

assert(differ(g, 4, gx, 4) == 4); //match up to the end

assert(differ(g, 4, gx, 2) == 2); //match up to the end but one is shorter

assert(differ(g, 9, gx, 9) == 8); //basic test until they differ

assert(differ(g, 0, gx, 9) == -1); //one input is nothing (nothing to compare)

assert(differ(g, 9, gx, 0) == -1); //one input is nothing (nothing to compare) string ga[2] = { "august", "august" };

string gxa[3] = { "august", "amanda", "joe" };

assert(differ(ga, 2, gxa, 3) == 1); //different sized arrays

assert(differ(ga, 2, gxa, -1) == -1); // negative input is bad

subsequence

string h[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

string hx[3] = { "diana", "tony", "" };

assert(subsequence(h, 9, hx, 3) == 2); // basic test

assert(subsequence(h, 9, hx, -2) == -1); // negative input is bad

assert(subsequence(h, 4, hx, 3) == -1); //cuts off before subsequence finishes

assert(subsequence(h, 5, hx, 3) == 2); //same as test above, but the subsequence just barely fits in

assert(subsequence(h, 5, hx, 0) == 0); //a subsequence of nothing starts at 0 as per the spec

assert(subsequence(h, 0, hx, 3) == -1); //if n1 is 0, error

lookupAny

string i[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

string ix[5] = { "sam", "bart", "don", "smith", "Tony" };

assert(lookupAny(i, 9, ix, 5) == -1); //makes sure it differentiates capitalization

string ia[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

string ixa[5] = { "sam", "bart", "don", "smith", "tony" };

assert(lookupAny(ia, 9, ixa, 5) == 3); //same as above but tony matches this time

assert(lookupAny(ia, 0, ixa, 5) == -1); //looking up nothing means nothing matches

assert(lookupAny(ia, 9, ixa, 0) == -1); //looking up something referencing nothing means nothing matches

string ib[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

string ixb[5] = { "sam", "selina", "don", "smith", "tony" };

assert(lookupAny(ib, 9, ixb, 5) == 0); //makes sure the first occurance is reported

split

string j[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(split(j, 9, "peter") == 4); //test with the splitter being in the array

assert(split(j, 9, "august") == 1); //test with the splitter being in the array

assert(split(j, 9, "zzzz") == 9); //return n because no elements are not < splitter

assert(split(j, 9, "tim") == 8); //basic test

assert(split(j, 9, "") == 0); //basic test

assert(split(j, 9, "diana") == 2); //basic test

assert(split(j, 0, "august") == 0); //no such elements in a blank array not less than august, so return n

string ja[9] = { "selina", "reed", "diana", "tony", "", "logan", "peter", "selina", "august" };

assert(split(ja, 3, "frank") == 1); //partial split test

cout << "All tests succeeded" << endl;