Angela Kan

Project 4 Report

1. The one notable obstacle I overcame was while coding the subsequence function: for me, it was definitely the trickiest one by far to figure out the code for, as I had to implement a lot of checking to make sure the arrays were only being accessed within the bounds specified by n1 and n2 while still checking for the entire a2 subsequence in a1. I did this eventually with an if statement and a bit of reasoning. I also ran into a problem for subsequence in which I was returning -1 too soon and didn’t check the entire a1 array, but I rectified this by using a boolean isSubseq instead of returning -1 immediately.
2. Test Cases:

appendToAll:

string g[4] = { "martha", "mark", "lindsey", "sara" };

n=full array size: assert(appendToAll(g, 4, "?") == 4 && g[0] == "martha?" && g[3] == "sara?");

string g1[4] = { "martha", "mark", "lindsey", "sara" };

n<full array size: assert(appendToAll(g1, 3, "3") == 3 && g1[0] == "martha3" && g1[3] == "sara");

string g2[4] = { "martha", "mark", "lindsey", "sara" };

n = 0: assert(appendToAll(g2, 0, "?") == 0 && g2[0] == "martha" && g2[3] == "sara");

string g3[4] = { "martha", "mark", "lindsey", "sara" };

n = negative assert(appendToAll(g3, -4, "?") == -1 && g3[0] == "martha" && g3[3] == "sara");

string is multiple characters: string g5[5] = { "donald", "joe", "mike", "lindsey", "kamala" };

assert(appendToAll(g5, 5, "!!!") == 5 && g5[0] == "donald!!!" && g5[4] == "kamala!!!");

lookup:

string h[7] = { "martha", "mark", "joe", "joe", "", "kamala", "lindsey" };

empty string among array: assert(lookup(h, 7, "kamala") == 5);

n=full array size, string present: assert(lookup(h, 7, "joe") == 2);

string not present in n-specified array: assert(lookup(h, 2, "joe") == -1);

n=0: assert(lookup(h, 0, "martha") == -1);

n=negative: assert(lookup(h, -2, "joe") == -1);

looking for empty string: assert(lookup(h, 7, "") == 4);

positionOfMax:

string k[6] = { "jamie", "lindsey", "mark", "susan", "joe", "donald" };

n = full array size: assert(positionOfMax(k, 6) == 3);

string k1[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

empty string present in array: assert(positionOfMax(k1, 7) == 3);

string k2[0] = { };

n = 0: assert(positionOfMax(k2, 0) == -1);

string k3[0] = { };

n = negative: assert(positionOfMax(k3, -5) == -1);

string k5[7] = { "", "", "", "susan", "", "kamala", "lindsey" };

all elements equal: assert(positionOfMax(k5, 3) == 0);

string k6[7] = { "", "", "", "susan", "", "kamala", "lindsey" };

empty string present in array, n<full array size: assert(positionOfMax(k6, 4) == 3);

string k7[7] = { "susan", "mark", "joe", "martha", "", "kamala", "lindsey" };

regular test case w/ position of max = 0: assert(positionOfMax(k7, 7) == 0);

string k8[7] = { "martha", "susan", "joe", "susan", "", "kamala", "lindsey" };

regular test case w/ empty string: assert(positionOfMax(k8, 7) == 1);

string k9[1] = { "martha"};

one element in array: assert(positionOfMax(k9, 1) == 0);

string k10[7] = { "susan", "mark", "joe", "martha", "", "kamala", "lindsey" };

n=0: assert(positionOfMax(k10, 0) == -1);

rotateLeft:

string l[5] = { "kamala", "jamie", "lindsey", "sara", "mark" };

n=full array size: assert(rotateLeft(l, 5, 1) == 1 && l[1] == "lindsey" && l[4] == "jamie");

string l3[0] = { };

n=0: assert(rotateLeft(l3, 0, 3) == -1);

pos = negative: assert(rotateLeft(l3, 0, -1) == -1);

n=negative: assert(rotateLeft(l3, -1, 3) == -1);

pos>n: assert(rotateLeft(l3, 0, 3) == -1);

string l4[4] = { "martha", "mark", "lindsey", "sara" };

n=0: assert(rotateLeft(l4, 0, 1) == -1);

pos = negative: assert(rotateLeft(l4, 4, -1) == -1);

pos = n: assert(rotateLeft(l4, 4, 4) == -1);

pos>n by 1: assert(rotateLeft(l4, 4, 5) == -1);

n=full array size, pos = 0 (entire array rotated): assert(rotateLeft(l4, 4, 0) == 0 && l4[0] == "mark" && l4[3] == "martha");

string l5[4] = { "martha", "", "lindsey", "sara" };

empty string in array: assert(rotateLeft(l5, 4, 1) == 1 && l5[1] == "lindsey" && l5[3] == "");

string l6[1] = { "martha" };

one element array: assert(rotateLeft(l6, 1, 0) == 0 && l6[0] == "martha");

string l7[4] = { "martha", "mark", "lindsey", "sara" };

pos = last element in array (nothing changes): assert(rotateLeft(l7, 4, 3) == 3 && l7[0] == "martha" && l7[3] == "sara");

pos = n: assert(rotateLeft(l7, 3, 3) == -1);

n<full array size: assert(rotateLeft(l7, 3, 2) == 2 && l7[0] == "martha" && l7[2] == "lindsey" && l7[3] == "sara");

n<full array size, different position than test case 1 above: assert(rotateLeft(l7, 3, 1) == 1 && l7[0] == "martha" && l7[1] == "lindsey" && l7[2] == "mark" && l7[3] == "sara");

countRuns:

string d[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "mike" };

array ending with 2 in a row: assert(countRuns(d, 9) == 5);

string d1[5] = { "mark", "mark", "mark", "susan", "susan" };

array with no 1-element sequences: assert(countRuns(d1, 5) == 2);

string d3[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "" };

array ending with empty string: assert(countRuns(d3, 9) == 6);

string d4[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "rob" };

array ending with 1-element run: assert(countRuns(d4, 9) == 6);

string d5[7] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe" };

array ending with 3-element run: assert(countRuns(d5, 7) == 4);

string d6[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "mike" };

n<full array size: assert(countRuns(d6, 4) == 3);

string d7[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "mike" };

n< full array size, ending w/ 1-element run: assert(countRuns(d7, 5) == 4);

string d8[0] = { };

n=0 and empty array: assert(countRuns(d8, 0) == 0);

string d9[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "mike" };

n=0 and non-empty array: assert(countRuns(d9, 0) == 0);

string d10[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "mike" };

n=negative; assert(countRuns(d10, -4) == -1);

string d11[9] = { "susan", "donald", "mike", "mike", "joe", "joe", "joe", "", "" };

array ending with 2-element empty string run: assert(countRuns(d11, 9) == 5);

flip:

string f[3] = { "lindsey", "joe", "mike" };

n=full array size: assert(flip(f, 3) == 3 && f[0] == "mike" && f[2] == "lindsey");

string f1[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

n<full array size: assert(flip(f1, 4) == 4 && f1[0] == "susan" && f1[1] == "" && f1[2] == "mike");

string f2[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

n=0: assert(flip(f2, 0) == 0 && f2[0] == "donald" && f2[1] == "mike");

string f3[0] = { };

empty array: assert(flip(f3, 0) == 0);

string f4[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

n=negative: assert(flip(f4, -2) == -1);

string f5[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

regular test case where n=full array size: assert(flip(f5, 6) == 6 && f5[0] == "jamie" && f5[1] == "sara" && f5[2] == "susan");

string f6[3] = { "lindsey", "joe", "mike" };

regular test case where n<full array size: assert(flip(f6, 2) == 2 && f6[0] == "joe" && f6[2] == "mike");

string f7[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

regular test case where n<full array size and empty string present in array: assert(flip(f7, 5) == 5 && f7[0] == "sara" && f7[1] == "susan" && f7[2] == "");

differ:

string mx[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

string my[4] = { "martha", "mark", "lindsey", "sara" };

n1<full array size: assert(differ(mx, 4, my, 4) == 2);

both n1 and n2 = full array size: assert(differ(mx, 7, my, 4) == 2);

n1 = 0: assert(differ(mx, 0, my, 4) == 0);

n1= negative: assert(differ(mx, -1, my, 4) == -1);

n2 = negative: assert(differ(mx, 4, my, -4) == -1);

n1 and n2 = 0: assert(differ(mx, 0, my, 0) == 0);

string mx1[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

string my1[5] = { "donald", "mike", "jamie", "", "susan" };

empty strings present in arrays: assert(differ(mx1, 6, my1, 5) == 2);

both n1 and n2 < full array size: assert(differ(mx1, 2, my1, 1) == 1);

string mx2[0] = { };

string my2[5] = { "donald", "mike", "jamie", "", "susan" };

empty a1 array: assert(differ(mx2, 0, my2, 5) == 0);

string mx3[5] = { "donald", "mike", "jamie", "susan", "sara" };

string my3[5] = { "donald", "mike", "jamie", "susan", "sara" };

identical arrays: assert(differ(mx3, 5, my3, 5) == 5);

subsequence:

string nx[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

string ny[4] = { "joe", "susan", "", "kamala" };

both n1 and n2 = full array size + empty string: assert(subsequence(nx, 7, ny, 4) == 2);

both n1 and n2 < full array size + empty string present: assert(subsequence(nx, 6, ny, 3) == 2);

a2 not present in a1 because a1 is too short: assert(subsequence(nx, 5, ny, 4) == -1);

string nx1[10] = { "kamala", "mark", "sara", "martha", "donald", "lindsey" };

string ny1[10] = { "mark", "sara", "martha" };

regular test case where a2 is present in a1: assert(subsequence(nx1, 6, ny1, 3) == 1);

string ny2[10] = { "kamala", "martha" };

a2 not present in a1: assert(subsequence(nx, 5, ny2, 2) == -1);

n1 = 0, n2 = negative: assert(subsequence(nx1, 0, ny1, -3) == -1);

n1 and n2 = 0: assert(subsequence(nx1, 0, ny1, 0) == 0);

n2 = 0: assert(subsequence(nx1, 9, ny1, 0) == 0);

n1 = 0: assert(subsequence(nx1, 0, ny1, 1) == -1);

n2>n1: assert(subsequence(nx1, 2, ny1, 3) == -1);

string nx3[10] = { "kamala", "mark", "liz", "martha", "donald", "lindsey" };

string ny3[10] = { "mark", "sara", "martha" };

a2 not present in a1: assert(subsequence(nx3, 6, ny3, 3) == -1);

string nx4[10] = { "kamala", "mark", "sara", "martha", "donald", "lindsey" };

string ny4[10] = { "donald", "lindsey", "martha" };

a2 is present in n1 but in different order: assert(subsequence(nx4, 6, ny4, 3) == -1);

string nx5[10] = { "kamala", "mark", "sara", "kamala", "mark", "lindsey" };

string ny5[10] = { "kamala", "mark", "lindsey" };

n-specified part of a2 present in a1: assert(subsequence(nx5, 6, ny5, 2) == 0);

a2 present in a1 after only part of a2 is present in a1: assert(subsequence(nx5, 6, ny5, 3) == 3);

string nx6[10] = { "kamala", "mark", "sara", "kamala", "mark", "lindsey" };

string ny6[10] = { "kamala", "sara", "kamala" };

a2 present in a1 but not consecutively: assert(subsequence(nx6, 6, ny6, 3) == -1);

lookupAny:

string ox[6] = { "kamala", "mark", "sara", "martha", "donald", "lindsey" };

string oy[4] = { "jamie", "donald", "martha", "mark" };

n1 and n2 = full array size: assert(lookupAny(ox, 6, oy, 4) == 1);

string oy1[10] = { "susan", "joe" };

no match: assert(lookupAny(ox, 6, oy1, 2) == -1);

string ox2[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

string oy2[3] = { "lindsey", "joe", "mike" };

multiple matches, check that earliest pos returned: assert(lookupAny(ox2, 7, oy2, 3) == 2);

n2 = 0: assert(lookupAny(ox2, 3, oy2, 0) == -1);

n1 and n2 = 0: assert(lookupAny(ox2, 0, oy2, 0) == -1);

n1 = negative: assert(lookupAny(ox2, -1, oy2, 3) == -1);

n2 < full array size: assert(lookupAny(ox2, 7, oy2, 1) == 6);

a2 not present in a1, both n1 and n2<full array size: assert(lookupAny(ox2, 6, oy2, 1) == -1);

string ox3[7] = { "", "mark", "joe", "susan", "", "kamala", "lindsey" };

string oy3[3] = { "lindsey", "", "mike" };

empty string match: assert(lookupAny(ox3, 7, oy3, 3) == 0);

string ox4[7] = { "", "mark", "joe", "susan", "", "kamala", "lindsey" };

string oy4[3] = { "dory", "rath", "amy" };

no matches: assert(lookupAny(ox4, 7, oy4, 3) == -1);

divide:

string p[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

given test case: assert(divide(p, 7, "lindsey") == 3);

n=0: assert(divide(p, 0, "lindsey") == 0);

string p1[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

one element array: assert(divide(p1, 1, "lindsey") == 0);

n = negative: assert(divide(p1, -11, "lindsey") == -1);

string p3[6] = { "jamie", "lindsey", "mark", "susan", "joe", "donald" };

n=full array size: assert(divide(p3, 6, "kamala") == 3);

string p4[4] = { "mark", "sara", "lindsey", "mike" };

n= full array size, divider = one of array elements: assert(divide(p4, 4, "mike") == 2);

string p5[4] = { "mark", "sara", "lindsey", "mike" };

n<full array size: assert(divide(p5, 3, "mike") == 2 && p5[3] == "mike");

string p6[3] = { "a", "a", "a" };

all elements in array are =divider: assert(divide(p6, 3, "a") == 0);

string p7[3] = { "a", "a", "a" };

all elements in array < divider: assert(divide(p7, 3, "b") == 3);

string p8[3] = { "b", "b", "b" };

all elements in array >divider: assert(divide(p8, 3, "a") == 0);

string p9[3] = { "c", "b", "a" };

n=full array size, check for correct ordering: assert(divide(p9, 3, "c") == 2 && p9[0] == "a" && p9[1] == "b" && p9[2] == "c");

string p10[3] = { "c", "b", "a" };

n<full array size, check for correct ordering: assert(divide(p10, 2, "c") == 1 && p10[0] == "b" && p10[1] == "c" && p10[2] == "a");