1. Notable Obstacles

One of the obstacles I had was with the CountRuns. This was because I initially had issues with nested loops causing incorrect counting. I fixed this by simplifying it to a single loop that compared it with the previous element.

Another obstacle I had was with PositionOfMax. I had to fix the comparison logic to handle identical elements by adding proper handling of the n = 0 case to return -1.

2. Test Cases and Reasoning:

appendToAll Tests:

string arr1[5] = { "a", "b", "c", "d", "e" };

assert(appendToAll(arr1, 5, "!") == 5);

assert(arr1[0] == "a!" && arr1[4] == "e!");

assert(appendToAll(arr1, -1, "x") == -1);

assert(appendToAll(arr1, 0, "x") == 0);

Reasoning: Tests basic appending, negative size, empty array, and verifies array contents

lookup Tests:

string arr2[5] = { "kamala", "donald", "jd", "tim", "jill" };

assert(lookup(arr2, 5, "tim") == 3);

assert(lookup(arr2, 5, "joe") == -1);

assert(lookup(arr2, 3, "jill") == -1);

assert(lookup(arr2, -1, "tim") == -1);

assert(lookup(arr2, 0, "tim") == -1);

Reasoning: Tests finding existing element, non-existent element, partial array search, invalid size

positionOfMax Tests:

string arr3[6] = { "doug", "kamala", "melania", "usha", "gwen", "donald" };

assert(positionOfMax(arr3, 6) == 3);

string maxArr[4] = {"tim", "tim", "tim", "tim"};

assert(positionOfMax(maxArr, 4) == 0);

assert(positionOfMax(arr3, -1) == -1);

assert(positionOfMax(arr3, 0) == -1);

Reasoning: Tests finding max, identical elements, empty array, negative size

rotateLeft Tests:

string arr4[5] = { "kamala", "donald", "jd", "tim", "joe" };

assert(rotateLeft(arr4, 5, 1) == 1);

assert(arr4[1] == "jd" && arr4[4] == "donald");

assert(rotateLeft(arr4, -1, 1) == -1);

assert(rotateLeft(arr4, 5, -1) == -1);

assert(rotateLeft(arr4, 5, 5) == -1);

Reasoning: Tests rotation, array modification verification, invalid position/size

countRuns Tests:

string arr5[9] = { "melania", "doug", "gwen", "gwen", "jill", "jill", "jill", "gwen", "gwen" };

assert(countRuns(arr5, 9) == 5);

string arr5a[3] = { "tim", "tim", "tim" };

assert(countRuns(arr5a, 3) == 1);

string singleRun[1] = {"a"};

assert(countRuns(singleRun, 1) == 1);

Reasoning: Tests multiple runs, single run, single element, consecutive duplicates

flip Tests:

string arr6[6] = { "kamala", "doug", "", "jill", "jd", "donald" };

assert(flip(arr6, 4) == 4);

assert(arr6[0] == "jill" && arr6[3] == "kamala");

assert(flip(arr6, -1) == -1);

assert(flip(arr6, 0) == 0);

Reasoning: Tests array reversal, content verification, empty array, negative size

differ Tests:

string arr7a[6] = { "kamala", "doug", "", "jill", "jd", "donald" };

string arr7b[5] = { "kamala", "doug", "donald", "", "jd" };

assert(differ(arr7a, 6, arr7b, 5) == 2);

assert(differ(arr7a, 2, arr7b, 1) == 1);

assert(differ(arr7a, -1, arr7b, 5) == -1);

Reasoning: Tests arrays of different lengths, finding first difference, invalid size

subsequence Tests:

string arr8a[6] = { "kamala", "tim", "usha", "gwen", "donald", "jd" };

string arr8b[3] = { "tim", "usha", "gwen" };

assert(subsequence(arr8a, 6, arr8b, 3) == 1);

string arr8c[2] = { "kamala", "gwen" };

assert(subsequence(arr8a, 5, arr8c, 2) == -1);

assert(subsequence(arr8a, 6, arr8b, 0) == 0);

Reasoning: Tests finding subsequence, non-existent subsequence, empty subsequence

lookupAny Tests:

string arr9a[6] = { "kamala", "tim", "usha", "gwen", "donald", "jd" };

string arr9b[4] = { "donald", "melania", "gwen", "tim" };

assert(lookupAny(arr9a, 6, arr9b, 4) == 1);

string arr9c[2] = { "jill", "joe" };

assert(lookupAny(arr9a, 6, arr9c, 2) == -1);

Reasoning: Tests finding first match, no matches, multiple possibilities

separate Tests:

string sepArr[6] = {"joe", "joe", "joe", "joe", "joe", "joe"};

assert(separate(sepArr, 6, "joe") >= 0 && separate(sepArr, 6, "joe") <= 6);

string arr10[6] = { "doug", "kamala", "melania", "usha", "gwen", "donald" };

int pos = separate(arr10, 6, "joe");

assert(pos >= 0 && pos <= 6);

for (int i = 0; i < pos; i++)

assert(arr10[i] < "joe");

for (int i = pos; i < 6; i++)

assert(arr10[i] >= "joe");

Reasoning: Tests separation around value, all equal elements, verifies ordering