CS 31 – Project 4 – Jahan Kuruvilla Cherian 104436427

1. The main difficulties in project 4 were the function subsequence and separate function. To overcome the subsequence function, I realized that using a Boolean would make comparisons and the definition of checks simpler, as did I learn that we can create a for loop that simultaneously loops through two arrays. The separate function obstacle was overcome by first decrementing the i to make sure each element was always checked, but this created an infinite loop and so the fix came with the use of the counter variable, that breaks out of the loop if the counter reaches the value of n.
2. Test Cases:

AppendToAll function: test[3] = {“glenn”, “maggie”, “rick”}

1. Try n = 0, to make sure this isn’t read as an error.

appendToAll(test,0,”!!!”)

1. Try to append some random string to make sure it appends to all the elements and returns the value of n.
2. appendToAll(test,2,”!!!”)
3. Let n < 0, to make sure it returns -1 and recognizes the error.

appendToAll(test,-1,”!!!”)

LookUp function: test[5] = {“glenn”, “maggie”, “rick”, “glenn”, “carl”}

1. Try a normal case with n>0 and a valid string element that is in the array to make sure the right value is returned.

lookup(test, 3, “maggie”)

1. Try a case where an array test has multiple of the same strings that are equal to target, to ensure that the smallest position of the location of target in test is returned.

lookup(test, 5, “glenn”)

1. Try the case of n = 0, where the return value should be -1 not because its an error, but because an empty array won’t contain the string target by definition.

lookup(test, 0, “rick”)

1. Try finding “Rick” in an array containing “rick” to make sure they are not equal and -1 is returned.

lookup(test, 5, “RicK”)

PositionOfMax function: test[4] = { "maggie", "carl", "carol", "maggie"}

1. Try n = 0, which in this case is defining the array as empty, and therefore should return -1.

positionOfMax(test, 0)

1. Make an array with multiple of the same string such as “maggie” as the highest valued string and make sure the smallest position is returned.

positionOfMax(test, 4)

1. Make an array with all the same string elements to make sure that the smallest index value is still returned if they are all equal.

test1[3] = {“daryl”, “daryl”, “daryl”)

positionOfMax(test1,3)

1. Make n smaller than the actual size of the array to make sure that only elements in the array up to n are evaluated and those beyond are not.

positionOfMax(test, 2)

RotateLeft and RotateRight functions: characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

1. Make n = 0 which in this case will be defined as holding no elements and therefore an error. Both functions should return -1.

rotateLeft(characters,0, 2)

rotateRight(characters,0,3)

1. Try making the position (pos) greater than the value of n, and as such should return -1 as this is not possible.

rotateLeft(characters,3,4)

rotateRight(characters,2,5)

1. Use normal values of n (where n is less than or equal to array size) and a valid position number, to make sure that the array is rearranged correctly respective to the function, and make sure it returns the correct original position.

rotateLeft(characters,3,1)

rotateRight(characters,5,4)

Flip function: roles[6] = { "abraham", "tara", "", "daryl", "carol", "tyreese" };

1. Let n be less than 0, and make sure the value returned is -1 as those conditions are errors, but if n = 0 then the value returned should just be 0, because technically flipping an empty array is not an error, and so should return n.

flip(roles,0)

1. Use an even number of n and then an odd number of n to make sure that in both cases the elements are appropriately reversed.

flip(roles,2)

flip(roles,5)

1. Let n be less than array size, to make sure the strings in the position above n are unaffected but those bounded by n are appropriately reversed.

flip(roles,3)

Differ function: roles[6] = { "abraham", "tara", "", "daryl", "carol", "tyreese" };

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

1. Try with n1 or n2 as 0 which is not an error in this case and make sure that the output is zero.

differ(roles,0,group,2)

1. Try arrays where the first few elements are equal and we just define n1 and n2 over those elements so as to return the smaller of n1 or n2.

differ(roles,2,group,1)

1. Try a normal case wherein the return value shows the smallest position of the differing elements.

differ(roles,6,group,5)

Subsequence function: names[10] = { "sasha", "rick", "beth", "glenn", "bob", "michonne", “rick”, “beth”, “glenn” };

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

names2[10] = {“sasha”, “bob”};

names3[10] = {“michone”}

1. Try n1 or n2 as equal to 0, which then means that because an empty sequence is always a subsequence of another array even if that the function should return 0.

subsequence(names, 0, names1, 3)

1. Make a2 contain elements of a1 but not in continuous sequence so that -1 is returned.

subsequence(names, 6, names2, 2)

1. Make a1 contain multiple sequences of a1 such that we get the smallest position first.

subsequence(names, 9, names1, 3)

1. Make a2 have only one string and still check if it’s a subsequence of a1.

subsequence(names, 9, names3, 1)

LookUpAny function: names[10] = { "sasha", "rick", "beth", "glenn", "rick", "michonne" };

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

1. Let n1 or n2 be 0 which isn’t an error, but because neither a1 nor a2 will contain any of the elements, the result should be -1.

lookupAny(names, 6, set1, 0)

1. Let n2 be bigger than n1 and check for any similar strings in a1.

lookupAny(names, 3, set1, 4)

1. Make a1 contain multiple elements from a2 so that the lowest position is returned.

lookupAny(names, 6, set1, 4)

Separate function: cast[6] = { "maggie", "carl", "daryl", "rick", "michonne", "carol" }

1. Make the separator equal to an element in the string and check if the value returned is the position equal to the separator.

separate(cast, 6, “daryl”)

1. Try a normal case where we rearrange the string in respect to the separator.

separate(cast, 6, “glenn”)

1. Try using a separator with capitals to make sure that it is still considered lower than the normal strings.

separate(cast, 6, “Daryl”)

1. Let n = 0, this should result in the function returning 0, due to the fact that an array with no element does not contain the separator and thus must return n.

separate(cast,0,”sasha”)

1. Use a separator that is smaller than all the strings in the array such that the value returned is n.

Separate(cast,6,”abbie”)