Roxane Martin

Professor Smallberg

CS 31

4 November 2019

Project 4

1. My only major obstacle during this project was figuring out how to program the separate function without making a new array. Originally, I cycled through and made an array with all the elements less than the separator and then used that to hold those values while I reordered the rest of the array. My solution was to cycle through the array once and find the position of the last value that would be less than the separator in the ordered array and then cycle through the whole array again and find any elements that occur before that position but are greater than the separator string and switch them with an element that occurs after the separation that is less than the separator.
2. Test data:

string h[7] = { “mick”, “marie”, “lucie”, “roxy”, “”, “gordon”, “lindsey” };

assert(lookup(h, 7, “gordon”) == 5); - check that lookup returns correct position when array contains element

assert(lookup(h, 7, “fiona”) == -1); - check that lookup returns -1 when array does not contain element

assert(lookup(h, 2, “roxy”) == -1); - check that lookup returns -1 when array contains element but is OUTSIDE of ‘interesting elements’

assert(positionOfMax(h, 7) == 3); - check that positionOfMax returns correct position of maximum element

string g[4] = { “mick”, “marie”, “lindsey”, “nancy” };

assert(differ(h, 4, g, 4) == 2); - check that differ returns first element where two arrays differ

assert(appendToAll(g, 4, “?”) == 4 && g[0] == “mick?” && g[3] == “nancy?”); - check that appendToAll adds character to each element and returns n

assert(rotateLeft(g, 4, 1) == 1 && g[1] == “lindsey?”); - check that rotate left rotates all elements left and moves first element to last postion and returns n

string e[4] = { “fiona”, “rudy”, “”, “gordon” };

assert(subsequence(h, 7, e, 4) == -1);- check that subsequence returns -1 when array does not contain subsequence

string j[4] = { “roxy”, “”, “gordon”, “lindsey”};

assert(subsequence(h, 7, j, 4) == 3); );- check that subsequence returns position of start of subsequence when array does contain subsequence

string d[5] = { “mary”, “mary”, “mary”, “ally”, “ally” };

assert(countRuns(d, 5) == 2); - check that countRuns returns number of runs in array

string f[3] = { “lindsey”, “fiona”, “mike” };

assert(lookupAny(h, 7, f, 3) == 6); - check that lookupAny returns position (in 1st array) of first element in 2nd array that appears in 1st array

string k[3] = { “allison”, “becca”, “mike” };

assert(lookupAny(h, 7, k, 3) == 6); - check that lookupAny returns -1 if no elements in 2nd array appear 1st array

assert(flip(k, 3) == 3 && k[0] == “mike” && k[2] == “allison”); - check that flip returns n and flips all elements in array

assert(separate(h, 7, “patrick”) == 6 && h[6] == “roxy”); - check that separate returns position of first element after separation and reorders all elements in array according to separator

assert(lookup(h, -5, "patrick") == -1);

assert(differ(h, -5, "patrick") == -1);

assert(positionOfMax(h, -5) == -1);

assert(separate(h, -5, "patrick") == -1);

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

assert(countRuns(d, -5) == -1);

assert(subsequence(h, 7, j, -5) == -1);

assert(flip(k, -5) == -1);

assert(appendToAll(g, -5, “?”) == -1);

assert(lookupAny(h, 7, k, -5) == -1);

- check that any function returns -1 if n is negative