Edward Chu 504772148

Report on CS 31 Assignment 4

Notable obstacles that I overcame:

1. For the flip function, I was considering to whether if I needed to make two parts of the function: One for arrays with an odd number of elements and one for even number of elements. However, after I thought about it, I realized that I did not need to do so, as number of elements/2 rounds down when it is odd.
2. I tried to not use extra arrays for flip, rotateLeft and separate, which was the hardest for separate. My initial approach was to separate elements into less than, equal to or greater than the separator, which was not possible without extra arrays. In the end, I used another function, rotateLeft, in order to accomplish my task.

Test Cases for Functions:

Append to all:

string g[4] = { "jill", "hillary", "gary", "mindy" };

assert(appendToAll(g, 4, "?") == 4 && g[0] == "jill?" && g[3] == "mindy?"); //Normal

string j[2] = {"jill", "mandy"};

assert(appendToAll(g, -5, "") == -1); //Negative length to check

assert(appendToAll(g, 4, "") == 4 && j[0] == "jill" && j[1] == "mandy"); //Appending empty string

Lookup:

string people[6] = { "hillary", "gary", "donald", "jill", "evan", "gary" };

assert(lookup(people, -5, "evan") == -1); //Negative length to check

assert(lookup(people, 3, "evan") == -1); //evan not found in interesting elements

assert(lookup(people, 4, "gary") == 1); //Normal

assert(lookup(people, 3, "hillary") == 0); //Normal, first element

assert(lookup(people, 6, "gary") == 1); //repeated gary, return pos of first gary

Position of Max:

string cand[6] = { "jill", "hillary", "timothy", "tim", "timothy", "tim" };

assert(positionOfMax(cand, 6) == 2); //Normal, timothy repeated, return position of first

assert(positionOfMax(cand, 0) == 0); //No element to examine

assert(positionOfMax(cand, -5) == -1); //Negative length to check

Rotate Left:

string lames[4] = { "jill?", "hillary?", "gary?", "mindy?" };

assert(rotateLeft(lames, 4, 5) == -1); //Out of scope

assert(rotateLeft(lames, -4, 1) == -1); //Negative length

assert(rotateLeft(lames, 4, 1) == 1 && lames[0] == "jill?" && lames[1] == "gary?" && lames[2] == "mindy?" && lames[3] == "hillary?"); //Normal

string games[4] = { "jill", "hillary", "gary", "mindy" };

assert(rotateLeft(games, 4, 3) == 3 && games[0] == "jill" && games[1] == "hillary" && games[2] == "gary" && games[3] == "mindy"); //Pos at the end of the array, should be no change

assert(rotateLeft(games, 4, 0) == 0 && games[0] == "hillary" && games[1] == "gary" && games[2] == "mindy" && games[3] == "jill"); //Pos at start of array

CountRuns:

string d[9] = {

"tim", "ajamu", "mike", "mike", "donald", "donald", "donald", "mike", "mike"

};

assert(countRuns(d, 9) == 5); //Normal

assert(countRuns(d, -9) == -1); //Negative length

string fames[2] = {"gary", "gary"};

assert(countRuns(fames, 2) == 1); //All elements in array same

assert(countRuns(fames, 0) == 0); //No elements examined, should be 0

Flip:

string f[3] = { "gary", "donald", "mike" };

assert(flip(f, 3) == 3);

assert(flip(f, -3) == -1); //Negative length

assert(f[0] == "mike" && f[2] == "gary" && f[1] == "donald"); //Normal, odd num of elements

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

assert(f[0] == "mike" && f[2] == "gary" && f[1] == "donald"); //No interesting elements

string k[4] = { "gary", "donald", "mike", "james"};

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

assert(k[0] == "james" && k[1] == "mike" && k[2] == "donald" && k[3] == "gary"); //Normal, even num of elements

Differ:

string folks[6] = { "ajamu", "mike", "", "tim", "mindy", "bill" };

string group[5] = { "ajamu", "mike", "bill", "", "tim" };

assert(differ(folks, -6, group, 5) == -1 && differ(folks, 6, group, -5) == -1); //Negative lengths

assert(differ(folks, 6, group, 5) == 2); //Normal

assert(differ(folks, 2, group, 1) == 1); //Normal, second array part of first array

string pple[7] = { "ajamu", "mike", "", "tim", "mindy", "bill", "lol" };

assert(differ(folks, 6, pple, 7) == 6); //First array part of second array

assert(differ(folks, 6, pple, 5) == 5); //Second array part of first array

string ppl[2] = { "Bob", "Jack"};

assert(differ(folks, 2, ppl, 1) == 0); //No elements equal

Subsequence:

string names[10] = { "evan", "hillary", "mindy", "jill", "ajamu", "gary" };

string names1[10] = { "hillary", "mindy", "jill" };

assert(subsequence(names, -6, names1, 3) == -1 && subsequence(names, 6, names1, -3) == -1 ); //Negative lengths

assert(subsequence(names, 6, names1, 3) == 1); //Normal

string names2[10] = { "evan", "jill" };

assert(subsequence(names, 5, names2, 2) == -1); //Not found

string names3[10] = { "ajamu", "gary", "jill"};

assert(subsequence(names, 5, names3, 3) == -1); //Not found, second array part of end of first array

LookupAny:

string names[10] = { "evan", "hillary", "mindy", "jill", "ajamu", "gary" };

string set1[10] = { "bill", "ajamu", "jill", "hillary" };

assert(lookupAny(names, 6, set1, 4) == 1); //Normal

assert(lookupAny(names, -6, set1, 4) == -1 && lookupAny(names, 6, set1, -4) == -1); //Negative lengths

string set2[10] = { "tim", "donald" };

assert(lookupAny(names, 6, set2, 2) == -1); //Not found

assert(lookupAny(names, 6, set2, 0) == -1); //Second array empty, should not be found

assert(lookupAny(names, 0, set2, 2) == -1); //First array empty, should not be found

Separate:

string cand2[6] = { "donald", "jill", "hillary", "tim", "evan", "bill" };

assert(separate(cand2, 6, "gary") == 3); //Normal

string h[7] = { "jill", "hillary", "donald", "tim", "", "evan", "gary" };

assert(separate(h, 7, "gary") == 3); //Normal with empty string

string cand3[4] = { "gary", "hillary", "jill", "donald" };

assert(separate(cand3, 4, "hillary") == 2); //Normal with element equal to separator

string cand4[4] = { "gary", "hillary", "hillary", "donald" };

assert(separate(cand4, 4, "hillary") == 2); //Multiple elements equal to separator

string cand5[4] = { "hillary", "hillary", "hillary", "hillary" };

assert(separate(cand5, 4, "hillary") == 4); //All elements equal to separator, no element less than separator

string cand6[4] = { "hillary", "hillary", "jill", "hillary" };

assert(separate(cand6, 4, "hillary") == 4); //All elements equal or greater than separator