**Chelsea Biala**

**CS31 Smallberg**

**Fall 2020**

1. For this project, I mostly struggled with the return statement, since the circumstances were different for each function. I also struggled with the divide function in the case where the divide was included within the array, because I could never get it quite in the middle. However, I’m definitely getting the hang of the debugger, because this helped me a lot in perfecting my code. (I know it doesn’t matter right now, but I do need to work on efficiency.)

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

string a2[6] = { "martha", "mark", "joe", "susan", "", "kamala" };

string a3[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

string a4[5] = { "joe", "susan", "", "kamala", "lindsey" };

string a5[2] = { "martha", "joe" };

assert(lookupAny(a, 7, a4, 5) == 2);

*for when a1 contains an element of a2*

assert(subsequence(a, 7, a5, 2) == -1);

*for when a1 contains a2 but out of order*

assert(subsequence(a, 7, a4, 5) == 2);

*for when a1 contains all of a2 in order*

assert(differ(a, 7, a2, 6) == 6);

*for when a1 contains all of a2 but a2 is smaller*

assert(differ(a, 7, a3, 7) == 7);

*for when a1 contains all of a2 and their sizes are equal*

assert(lookup(a, 7, "mark") == 1);

*for when the array contains the argument*

assert(lookup(a, 7, "chelsea") == -1);

*for when the array doesn’t contain the argument*

assert(lookup(a, 7, "Mark") == -1);

*for when the array contains the argument but with different capitalization*

assert(positionOfMax(a, 7) == 3);

*for when max is present*

assert(countRuns(a, 7) == 7);

*for when all elements are unique*

assert(rotateLeft(a, 7, 6) == 6 && a[0] == "martha" && a[6] == “lindsey");

*for when you rotate the last element*

assert(rotateLeft(a, 7, 0) == 0 && a[0] == "mark" && a[6] == “martha");

*for when you rotate the first element*

string a6[5] = { "joe", "hi", "", "kamala", "lindsey" };

assert(divide(a6, 5, "hi") == 1 && a6[0] == "" && a6[2] > “hi");

*for when the dividing element is in the array*

string b[3] = {"chel", "izzy", "yui"};

assert(subsequence(a, 7, b, 3) == -1);

*for when a1 does not contain any of a2*

assert(rotateLeft(b, 3, 1) == 1 && b[0] == "chel" && b[1] == "yui" && b[2] == “izzy");

*for when you rotate the second element*

assert(appendToAll(b, 3, "!") == 3 && b[0] == "chel!" && b[1] == "yui!" && b[2] == “izzy!");

*for when you append to all in general*

assert(flip(b, 3) == 3 && b[0] == "izzy!" && b[1] == "yui!" && b[2] == “chel!");

*for when you flip an array with an odd number of elements*

assert(lookupAny(a, 7, b, 3) == -1);

*for when a1 contains none of a2*

string c[4] = {"hi", "bye", "hi", "bye"};

string c2[2] = {"hi", "bye"};

assert(subsequence(c, 4, c2, 2) == 0);

*for when a1 contains a2 more than once*

assert(lookupAny(c, 4, c2, 2) == 0);

*for when a1 contains a2 more than once*

assert(lookup(c, 4, "bye") == 1);

*for when the array contains the argument*

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

*for when there is more than one max*

assert(flip(c, 4) == 4 && c[0] == "bye" && c[1] == "hi" && c[2] == "bye" && c[3] == “hi");

*for when you flip an array with an even number of elements*

string d[1] = {"chelsea"};

assert(rotateLeft(d, 1, 0) == 0);

*for when you rotate an array with one element*

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

*for when there’s an array with one unique element*

assert(divide(d, 1, "a") == 0);

*for when the divider is less than all elements*

assert(divide(d, 1, "zoo") == 1);

*for when the divider is greater than all elements*

string e[5] = {"", "", "hi", "", ""};

assert(countRuns(e, 5) == 3);

*for when there are multiple identical runs*

string f[0];

assert(appendToAll(f, 0, "!") == 0);

assert(lookup(f, 0, "") == -1);

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

assert(rotateLeft(f, 0, -2) == -1);

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

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

assert(differ(e, 5, f, 0) == 0);

assert(subsequence(e, 5, f, 0) == 0);

assert(subsequence(f, 0, e, 5) == -1);

assert(lookupAny(e, 5, f, 0) == -1);

assert(lookupAny(f, 0, e, 5) == -1);

assert(divide(f, 0, "!") == 0);

*all for when functions are called on arrays where n is 0*

string g[0];

assert(differ(f, 0, g, 0) == 0);

assert(subsequence(f, 0, g, 0) == 0);

*both for when functions called on two arrays where n is 0*

assert(appendToAll(b, -1, "!!!") == -1);

assert(lookup(b, -1, "mark") == -1);

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

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

assert(countRuns(b, -1) == -1);

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

assert(differ(b, -1, a, 0) == -1);

assert(subsequence(b, -1, a, 0) == -1);

assert(lookupAny(b, -1, a, 0) == -1);

assert(divide(b, -1, "!") == -1);

*all 10 above for when an invalid n is passed in*