Sadhana Vadrevu

CS 31

Professor Smallberg

Project 4 Report

The hardest part of this project for me was creating the divide function. Two specific issues I had with this function were that I kept on getting stuck in an infinite loop once all the terms greater than the divider were at the end of the array and I wasn’t sure how to deal with terms that were equal to the divider. At first, I kept on getting stuck in a continuous loop because I kept checking every term, even after they were in their correct position. However, after drawing out what I wanted the function to do, I realized that I needed to decrement the number of terms I was looking at each time I moved a term to the end of the array. This adjustment allowed the loop to not check the terms that were already in place. The other problem I ran into, dealing with terms equal to divider, was also easier to solve once I drew it out. I realized that the terms equal to the divider belong at the returnValue position and the positions following that (if there was more than one term in the array equal to the divider). After the terms were in their position, they did not need to be checked anymore. I made sure that they wouldn’t be by coding it into my else-if statement.

**Test Cases:**

**appendToAll Tests**

string h[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

assert(appendToAll(h, 5, "hi") == 5 && h[2] == "edhi" && h[4] == "hi" && h[6] == "fiona"); *// check that function works correctly*

assert (appendToAll(h, -3, "\*\*") == -1 && h[4] == "hi"); *// check that function handles a negative n correctly*

assert (appendToAll(h, 0, "\*\*") == 0 && h[5] == "eleni"); *// check that function handles n = 0 correctly*

**lookup Tests**

string j[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

assert(lookup(j, 7, "eleni") == 5); *// check that function works correctly*

assert(lookup(j, 7, "ed") == 2); *// check that function works correctly*

assert(lookup(j, 2, "ed") == -1); *// term is not within bounds of array*

assert(lookup(j, -4, "ed") == -1); *// checks that function handles a negative n correctly*

assert(lookup(j, 0, "ed") == -1); *// checks that functions handles n = 0 correctly*

**positionOfMax Tests**

assert(positionOfMax(j, 7) == 3); *// checks that function works correctly*

assert(positionOfMax(j, -4) == -1); *// checks that functions handles a negative n correctly*

assert(positionOfMax(j, 0) == -1); *// checks that functions handles n = 0 correctly*

string p[4] = { "ed" , "ed" , "ed"};

assert(positionOfMax(p, 2) == -1); *// checks that function handles array with no unique terms correctly*

**rotateLeft Tests**

string g[4] = { "greg", "gavin", "fiona", "kevin" };

assert(rotateLeft(g, 3, 1) == 1 && g[2] == "gavin" && g[1] == "fiona"); *// checks that function works correctly*

assert(rotateLeft(g, -3, 1) == -1); *// checks that functions handles a negative n correctly*

assert (rotateLeft(g, 3, -1) == -1); *// checks that functions handles a negative pos correctly*

assert(rotateLeft(g, 0, 1) == -1); *// checks that functions handles a impossible pos (pos >= n) correctly*

assert(rotateLeft(g, 0, 0) == -1); *// checks that functions handles a impossible pos (pos >= n) correctly*

assert (rotateLeft(g, 2, 2) == -1); *// checks that functions handles a impossible pos (pos >= n) correctly*

assert(rotateLeft(g, 4, 1) == 1 && g[1] == "gavin" && g[3] == "fiona"); *// checks that function works correctly*

**countRuns Tests**

string d[5] = { "gavin", "gavin", "gavin", "xavier", "xavier" };

assert(countRuns(d, 5) == 2); *// checks that function works correctly*

assert(countRuns(d, 2) == 1); *// checks that function works correctly*

assert(countRuns(d, 0) == 1); *// checks that function handles n = 0 correctly*

assert(countRuns(d, -4) == -1); *// checks that function handles a negative n correctly*

**flip Tests**

assert(flip(d, 4) == 4 && d[0] == "xavier" && d[3] == "gavin" && d[4] == "xavier"); *// checks that function works correctly*

assert(flip(d, 0) == 0 && d[0] == "xavier"); *// checks that function handles n = 0 correctly*

assert(flip(d, -3) == -1 && d[3] == "gavin"); *// checks that function handles a negative n correctly*

string f[3] = { "fiona", "ed", "john" };

assert(flip(f, 3) == 3 && f[0] == "john" && f[2] == "fiona"); *// checks that function works correctly*

**differ Tests**

string folks[6] = { "betty", "john", "", "xavier", "kevin", "dianne" };

string group[5] = { "betty", "john", "dianne", "", "xavier" };

assert(differ(folks, 6, group, 5) == 2); *// checks that function works correctly when arrays have a differing term*

assert(differ(folks, 2, group, 1) == 1); *// checks that function works correctly when arrays are equal*

assert(differ(folks, -1, group, 3) == -1); *// checks that function works correctly when array size is negative*

assert(differ(folks, 1, group, -3) == -1); *// checks that function works correctly when array size is negative*

assert(differ(folks, 0, group, 4) == 0); *// checks that function works correctly when array size is 0*

assert(differ(folks, 3, group, 0) == 0); *// checks that function works correctly when array size is 0*

**subsequence Tests**

string e[4] = { "ed", "xavier", "", "eleni" };

assert(subsequence(j, 7, e, 4) == 2); *// checks that function works correctly*

assert(subsequence(j, 4, e, 2) == 2); *// checks that function works correctly*

assert(subsequence(folks, 4, e, 2) == -1); *// checks that function works correctly*

assert(subsequence(j, -1, e, 2) == -1); *// checks that function works correctly when array size is negative*

assert(subsequence(j, 4, e, -1) == -1); *// checks that function works correctly when array size is negative*

assert(subsequence(j, 0, e, 1) == -1); *// checks that function works correctly when array size is 0*

assert(subsequence(j, 4, e, 0) == 0); *// checks that function works correctly when array size is 0*

**lookupAny Tests**

assert(lookupAny(folks, 6, e, 4) == 2); *// checks that function works correctly*

assert(lookupAny(e, 2, folks, 2) == -1); *// checks that function works correctly when there is no match found*

assert(lookupAny(folks, 0, e, 3) == -1); *// checks that function works correctly when array size is 0*

assert(lookupAny(folks, 3, e, 0) == -1); *// checks that function works correctly when array size is 0*

assert(lookupAny(folks, -1, e, 3) == -1); *// checks that function works correctly when array size is negative*

assert(lookupAny(folks, 4, e, -3) == -1); *// checks that function works correctly when array size is negative*

**divide Tests**

assert(divide(j, 7, "fiona") == 3); *// checks that function works correctly*

assert(divide(e, 4, "sally") == 3 && e[3] == "xavier"); *// checks that function works correctly*

string b[5] = { "sue", "sue", "adam", "", "xavier"};

assert(divide(b, 5, "sue") == 2 && b[4] == "xavier" && b[3] == "sue"); *// checks that function works correctly when there are multiple terms equal to the divider*

assert(divide(b, -3, "") == -1); *// checks that function works correctly when array size is negative*