Project 4 Report

Siddarth Chalasani

# Obstacles Overcome

I found the divide and subsequence functions to be difficult to implement. Initially, I called the differ and rotateLeft functions in the code for subsequence. However, since rotateLeft changes the array, and subsequence treats the arrays in the arguments as constants, the program didn’t compile. Therefore, I had to find a way using multiple for loops to implement the subsequence function. For divide, some test cases would cause my program to go into an infinite loop, and it couldn’t handle the case when the divider was the empty string, I found that the reason it was going into an infinite loop was that I added i--; into a for loop, so it would increment and decrement i, leading to the program staying in the for loop forever, so I found a way to implement the function without i--;. I dealt with the case when the divider is zero by adding a condition that, when the divider is “”, divide would return 0.

# Test Cases

// test cases given by project spec

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

assert(lookup(h, 7, "eleni") == 5);

assert(lookup(h, 7, "ed") == 2);

assert(lookup(h, 2, "ed") == -1);

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

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

assert(differ(h, 4, g, 4) == 2);

assert(appendToAll(g, 4, "?") == 4 && g[0] == "greg?" && g[3] == "kevin?");

assert(rotateLeft(g, 4, 1) == 1 && g[1] == "fiona?" && g[3] == "gavin?");

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

assert(subsequence(h, 7, e, 4) == 2);

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

assert(countRuns(d, 5) == 2);

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

assert(lookupAny(h, 7, f, 3) == 2);

assert(flip(f, 3) == 3 && f[0] == "john" && f[2] == "fiona");

assert(divide(h, 7, "fiona") == 3);

// testing appendToAll

string appendToAll1[5] = { "dianne", "fiona", "ed", "xavier", "greg" };

assert(appendToAll(appendToAll1, 5, "!!!") == 5 && appendToAll1[0] == "dianne!!!");

// when n == 0, the array shouldn't be modified

string appendToAll2[5] = { "dianne", "fiona", "ed", "xavier", "greg" };

assert(appendToAll(appendToAll2, 0, "!!!") == 0 && appendToAll2[3] == "xavier");

// if value == 0, the array shouldn't be modified

assert(appendToAll(appendToAll2, 5, "") == 5 && appendToAll2[0] == "dianne");

// if n < 0, appendToAll should return -1 and the array shouldn't be modified

assert(appendToAll(appendToAll2, -1, "") == -1 && appendToAll2[1] == "fiona");

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

// testing lookup

// normal-functioning test case

assert(lookup(people, 7, "xavier") == 3);

// target isn't in string

assert(lookup(people, 7, "gregory") == -1);

// target's case is different

assert(lookup(people, 7, "XAVIER") == -1);

// target exists, but not within bounds

assert(lookup(people, 4, "eleni") == -1);

// n == 0

assert(lookup(people, 0, "ed") == -1);

// target is empty string and exists in array

assert(lookup(people, 7, "") == 4);

// target is empty string and is not in array

assert(lookup(people, 3, "") == -1);

// n < 0

assert(lookup(people, -3, "") == -1);

// testing positionOfMax

// normal-functioning test cases

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

assert(positionOfMax(people, 3) == 0);

// ties

string eds[7] = { "ed", "ed", "ed", "ed", "ed", "ed", "ed" };

assert(positionOfMax(eds, 3) == 0);

// n == 0

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

// n < 0

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

// Now we move on to rotateLeft

string rotate1[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

// normal-functioning test case

assert(rotateLeft(rotate1, 7, 1) == 1 && rotate1[1] == "ed" && rotate1[6] == "gavin");

// pos refers to last element in array, so array should be unchanged

assert(rotateLeft(rotate1, 7, 6) == 6 && rotate1[1] == "ed" && rotate1[6] == "gavin");

// n == 0, so the array should be unchanged

assert(rotateLeft(rotate1, 0, 0) == -1 && rotate1[1] == "ed" && rotate1[6] == "gavin");

// n < 0, so array should be unchanged

assert(rotateLeft(rotate1, -1, 1) == -1 && rotate1[1] == "ed" && rotate1[6] == "gavin");

// pos < 0, so array should be unchanged

assert(rotateLeft(rotate1, 1, -1) == -1 && rotate1[1] == "ed" && rotate1[6] == "gavin");

// pos >= 0, so array should be unchanged

assert(rotateLeft(rotate1, 7, 7) == -1 && rotate1[1] == "ed" && rotate1[6] == "gavin");

// testing countRuns

// when all elements are the same

assert(countRuns(eds, 7) == 1);

// all elements are different

string count1[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

assert(countRuns(count1, 7) == 7);

// same elements, but not all are consecutive

string count2[9] = { "xavier", "betty", "john", "john", "ed", "ed", "ed", "john", "john" };

assert(countRuns(count2, 9) == 5);

// all elements have different combinations of upper and lower case

string caseGregs[7] = { "greg", "GREG", "greG", "gReg", "grEg", "GreG", "gREg" };

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

// n == 0

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

// n < 0

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

// testing flip

// normal-functioning test case and checking if bounds are respected (odd numbered flip)

string flip1[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

assert(flip(flip1, 3) == 3 && flip1[0] == "ed" && flip1[1] == "gavin" && flip1[2] == "greg" && flip1[4] == "");

// even numbered flip

string flip2[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

assert(flip(flip2, 4) == 4 && flip2[0] == "xavier" && flip2[2] == "gavin");

// n == 0, so array remains unchanges

assert(flip(flip2, 0) == 0 && flip2[0] == "xavier" && flip2[2] == "gavin");

// n < 0, so array remains unchanged

assert(flip(flip2, -1) == -1 && flip2[0] == "xavier" && flip2[2] == "gavin");

//testing differ

string differ1[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

string differ2[7] = { "greg", "gavin", "ed", "xavier", "d", "eleni", "fiona" };

string differ3[7] = { "", "gavin", "ed", "xavier", "d", "eleni", "fiona" };

// normal-functioning case

assert(differ(differ1, 7, differ2, 7) == 4);

// one is smaller than the other

assert(differ(differ1, 7, differ2, 4) == 4);

// both are exactly the same

assert(differ(differ1, 7, differ1, 7) == 7);

// first element is not the same

assert(differ(differ1, 7, differ3, 7) == 0);

// n1 or n2 == 0

assert(differ(differ1, 0, differ2, 7) == 0);

// n1 or n2 < 0

assert(differ(differ1, -1, differ2, 7) == -1);

// testing subsequence

string subsequence1[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

string subsequence2[4] = { "ed", "xavier", "", "eleni" };

string subsequence3[4] = { "greg", "greg", "greg", "greg" };

// normal-functioning case

assert(subsequence(subsequence1, 7, subsequence2, 4) == 2);

// not a subsequence

assert(subsequence(subsequence1, 7, subsequence3, 4) == -1);

// n2 == 0

assert(subsequence(subsequence1, 7, subsequence3, 0) == 0);

// n1 == 0

assert(subsequence(subsequence1, 0, subsequence3, 3) == -1);

// n1 or n2 < 0

assert(subsequence(subsequence1, -1, subsequence3, 0) == -1);

// testing lookupAny

string lookupAny1[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

string lookupAny2[3] = { "greg", "matt", "allison" };

string lookupAny3[3] = { "leo", "matt", "allison" };

string lookupAny4[7] = { "greg", "gavin", "", "xavier", "", "eleni", "fiona" };

string lookupAny5[3] = { "", "matt", "allison" };

// standard case

assert(lookupAny(lookupAny1, 7, lookupAny2, 3) == 0);

// no matching elements

assert(lookupAny(lookupAny1, 7, lookupAny3, 3) == -1);

// matching string appears twice

assert(lookupAny(lookupAny4, 7, lookupAny5, 3) == 2);

// n2 == 0

assert(lookupAny(lookupAny1, 7, lookupAny3, 0) == -1);

// n1 == 0

assert(lookupAny(lookupAny1, 0, lookupAny3, 3) == -1);

// n1 or n2 < 0

assert(lookupAny(lookupAny1, -1, lookupAny3, 3) == -1);

// testing divide

string divide1[7] = { "greg", "gavin", "ed", "xavier", "matt", "eleni", "fiona" };

string divide2[7] = { "greg", "gavin", "ed", "xavier", "matt", "eleni", "fiona" };

// divider is not in array

assert(divide(divide1, 7, "p") == 6);

// divider is larger than everything

assert(divide(divide1, 7, "z") == 7);

// divider is empty string

assert(divide(divide1, 7, "") == 0);

// everyting is equal to divider

assert(divide(eds, 7, "ed") == 0);

// n < 0

assert(divide(eds, -1, "ed") == -1);

cout << "All tests succeeded" << endl;