Oo, Thant Zin (Andy)

CS 31, Section 2F

Project 4 Report

1. The main obstacles I had to overcome were the implementation of divide() and subsequence(). The stipulation that no additional arrays should be created in divide was tricky. Being able to create a temporary array would have made the implementation of divide easier by far. Instead, I used separate loops to keep track of how many strings were < divider, == divider and > divider. Although it works, I feel like there was a much more efficient way to implement the function. For subsequence, my main obstacle was wrapping my head around the spec. I was uncertain as to what the return value should be for cases in which n1 and/or n2 == 0. I had to reread Prof. Smallberg’s spec on sequences of 0 elements a couple times to understand what he meant.
2. Aside from the test cases provided in the Project 4 Spec testing for basic functionality, I wrote some test cases to check how my functions handled specific scenarios.

appendToAll():

string h[5] = { "andy", "siddarth", "oliver", "xavier", "ed" };

**When n is negative:**

assert(appendToAll(h, -1, "") == -1 && h[2] == "oliver");

**When** **string to append is empty:**

assert(appendToAll(h, 5, "") == 5 && h[2] == "oliver");

lookup():

string i[7] = { "andy", "siddarth", "oliver", "xavier", "ed", “”, “nolan” };

**Case sensitivity:**

assert(lookup(i, 7, "OLIVER") == -1);

**Nonexistent string:**

assert(lookup(i, 7, "jenn") == -1);

**When n = 0:**

assert(lookup(i, 0, "andy") == -1);

**Empty string:**

assert(lookup(i, 7, "") == -1);

**When n is negative:**

assert(lookup(i, -1, "andy") == -1);

positionOfMax():

**When n is negative:**

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

**When n = 0:**

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

rotateLeft():

**When pos = (n – 1), the values should be unchanged because the last string is selected.**

assert(rotateLeft(i, 7, 6) == 6 && i[1] == "siddarth" && i[6] == "nolan");

**When n = 0, the values should be unchanged.**

assert(rotateLeft(i, 0, 0) == -1 && i[1] == "siddarth" && i[6] == "nolan");

**When n is negative, the values should be unchanged.**

assert(rotateLeft(i, -1, 1) == -1 && i[1] == "siddarth" && i[6] == "nolan");

**When pos is negative, the values should be unchanged.**

assert(rotateLeft(i, 1, -1) == -1 && i[1] == "siddarth" && i[6] == "nolan");

**When pos >= n, the values should be unchanged.**

assert(rotateLeft(i, 7, 7) == -1 && i[1] == "siddarth" && i[6] == "nolan");

countRuns():

**When array i has no runs, should return n.**

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

**When array has inconsecutive runs of the same string.**

string consecutive[8] = { "andy", "andy", "siddarth", "oliver", "oliver", "vishal", "aria", "andy" };

assert(countRuns(consecutive, 8) == 6);

**When array handles case sensitive strings, should return n.**

string andyCase[5] = { "andy", "ANDY", "Andy", "aNdy", "andY" };

assert(countRuns(andyCase, 5) == 5);

**When n = 0.**

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

**When n is negative.**

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

flip():

**When n = 0.**

assert(flip(consecutive, 0) == 0 && consecutive[0] == "andy" && consecutive[2] == "siddarth");

**When n is negative.**

assert(flip(consecutive, -1) == -1 && consecutive[0] == "andy" && consecutive[2] == "siddarth");

differ():

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

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

**Normal case.**

assert(differ(differ1, 7, differ2, 7) == 2);

**When one is smaller than the other.**

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

**When given the same array and array size.**

assert(differ(differ2, 5, differ2, 5) == 5);

**When any n = 0.**

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

**When n is negative.**

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

subsequence():

string sub1[5] = { "greg", "gavin", "xavier", "", "eleni " };

string sub2[5] = { "ed", "xavier", "", "eleni", "fiona" };

**When there is no subsequence.**

assert(subsequence(sub1, 2, sub2, 5) == -1);

**When n2 = 0.**

assert(subsequence(sub1, 5, sub2, 0) == 0);

**When n1 = 0.**

assert(subsequence(sub1, 0, sub2, 5) == -1);

**When n is negative.**

assert(subsequence(sub1, -1, sub2, 0) == -1);

lookupAny():

**No similarities.**

assert(lookupAny(sub1, 2, sub2, 5) == -1);

**When n1 = 0.**

assert(lookupAny(sub1, 0, sub2, 5) == -1);

**When n2 = 0.**

assert(lookupAny(sub1, 5, sub2, 0) == -1);

**When n is negative.**

assert(lookupAny(sub1, -1, sub2, 4) == -1);

divide():

string divide[7] = { "greg", "gavin", "andy", "xavier", "oliver", "eleni", "fiona" };

**When divider isn’t in array (should still be valid input).**

assert(divide(divide, 7, "w") == 6);

**When divider is empty string.**

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

**When divider is larger than every element in array.**

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

**When n is negative.**

assert(divide(divide, -1, "andy") == -1);

(Test cases provided in Project 4 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);

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