Shannon Hoang

604787743

Proj 4 Report

1. Notable obstacles

Some notable obstacles were implementing makeMerger and divide. It was really hard to think about what logic would be used for divide until I realized that I could just use a function I had previously written, moveToBeginning.

Sometimes it wasn’t my actual functions that gave me trouble, but also realizing that the way some of my test cases were written were making it seem like the code was broken, when in reality it wasn’t. For example, if movetoend or movetobeginning is run twice in a row on the same string, the results that may be expected won’t be seen, causing you to think you’ve written something wrong.

An obstacle in general, was really questioning whether or not I had covered all the test cases correctly and looking through the scope really thoroughly. Because at least in my opinion, for this project, all the possible bad cases, weren’t as clear.

2. Test Cases

//test cases for ENUMERATE

string animals1[4] = {"dog", "cat", "dog", "dog"};

string dog = "dog";

string cat = "cat";

string bird = "bird";

//normal cases

assert(enumerate (animals1, 4, dog) == 3);

assert(enumerate (animals1, 4, cat) == 1);

//no elements equal to, should be 0

assert(enumerate (animals1, 4, bird) == 0);

//number of elements 0

assert(enumerate (animals1, 0, dog) == 0);

//neg number of elements

assert(enumerate (animals1, -5, bird) == -1);

cerr << "PASSED: enumerate" << endl;

//test cases for LOCATE

string animals2[4] = {"dog", "cat", "dog", "dog"};

//there are 3 dogs, it should return 0, because first one

assert(locate(animals2, 4, dog) == 0);

assert(locate(animals2, 4, cat) == 1);

//no element matches target, return -1

assert(locate(animals2, 4, bird) == -1);

//number of elements = 0, no element matches

assert(locate(animals2, 0, cat) == -1);

//negative number of elements

assert(locate(animals2, -1, cat) == -1);

cerr << "PASSED: locate" << endl;

// test case for locate sequence

string locateNames[9] = {

"lana", "cyril", "malory", "malory", "malory",

"sterling", "sterling", "malory", "malory"};

string target = "sterling";

int b;

int e;

//normal case

assert(locateSequence(locateNames, 9, "malory", b, e) == true);//return true, b=2 and e=3

//check if right beginning and end

assert(b == 2 && e == 4);

assert(locateSequence(locateNames, 9, "cyril", b, e) == true);//return true, b=1 and e=1

assert(b == 1 && e == 1);

// case where sequence ends at end of string

assert(locateSequence(locateNames, 4, "malory", b, e) == true);//return true, b=2 and e=3

assert(b == 2 && e == 3);

//no string equal to target

b = 0; e = 0;

assert(locateSequence(locateNames, 9, "petty", b, e) == false);

//number of elements = 0

assert(locateSequence(locateNames, 0, "malory", b, e) == false);

//negative number of elements

assert(locateSequence(locateNames, -3, "malory", b, e) == false);

assert(b == 0 && e == 0);//previously set, should be false,

cerr << "PASSED: locateSequence" << endl;

//locationOfMin test cases

string fruits[3] = {"kiwis", "clementines", "cantalope"};

assert (locationOfMin(fruits, 3) == 2);

string places[6] =

{"costa rica", "zealand", "california", "massachusetts","appalachians", "antartica" };

//costrica vs zealand

assert(locationOfMin(places, 2) == 0);

//first 3 elements

assert(locationOfMin(places, 3) == 2);

assert(locationOfMin(places, 6) == 5);

// neg/0 input

assert(locationOfMin(places, -1) == -1);

assert(locationOfMin(places, 0) == -1);

//two elements that are min

string stds[4] = {"herpes", "chlamydia", "crabs", "chlamydia"};

assert(locationOfMin(stds, 4) == 1);

// all elements are the same and are min

string feelings[2] = {"sad", "sad"};

assert(locationOfMin(feelings,2) == 0);

cerr << "PASSED: locationOfMin" << endl;

//moveToEnd tests

// move lana to the end

string ppl1[5] = { "malory", "lana", "sterling", "cheryl", "cyril" };

string ppl69[5] = {"malory", "lana", "sterling", "cheryl", "cyril"};

assert(moveToEnd(ppl1, 5, 1) == 1 && ppl1[4] == "lana");

for(int i = 1; i < 5 - 1;i++) {

assert(ppl1[i] == ppl69[i + 1]);

}

//move cyril to the end

string ppl2[5] = {"malory", "lana", "sterling", "cheryl", "cyril"};

assert (moveToEnd(ppl2, 5, 4) == 4 && ppl2[4] == "cyril");

//edge case: goes beyond scope of array

assert (moveToEnd(ppl1, 5, 6) == -1);

//edge case: negative element value

assert (moveToEnd(ppl1, -5, 1) == -1);

//edge case: n=0, examine no array elements but want to move smth

assert(moveToEnd(ppl1, 0, 0) == -1);

cerr << "PASSED: moveToEnd" << endl;

//moveToBeginning tests

// move cyril to beginning

string ppl3[5] = { "malory", "lana", "sterling", "cheryl", "cyril" };

string ppl70[5] = { "malory", "lana", "sterling", "cheryl", "cyril" };

moveToBeginning(ppl3, 5, 4);

for(int i = 0; i < 4; i++) {

assert(ppl3[i + 1] == ppl70[i]);

}

assert(ppl3[0] == "cyril");

//move lana to beginning

string ppl4[5] = { "malory", "lana", "sterling", "cheryl", "cyril" };

moveToBeginning(ppl4, 5, 1);

assert(ppl4[0] == "lana");

//move malory to the beginning

string ppl5[5] = { "malory", "lana", "sterling", "cheryl", "cyril" };

assert(moveToBeginning(ppl5, 5, 0) == 0);

assert(ppl5[0] == "malory");

//edge case: goes beyond array scope

assert(moveToBeginning(ppl5, 5, 5) == -1);

//edge case: negative number of elements

assert(moveToBeginning(ppl5, -5, 4) == -1);

//n=0, examine no array elements

assert(moveToBeginning(ppl5, 0, 0)== -1);

cerr << "PASSED: moveToBeginning" << endl;

//locate difference test cases

string cast[5] = { "malory", "lana", "sterling", "cheryl", "cyril" };

string roles[4] = { "malory", "lana", "krieger", "ray" };

//same up to sterling/krieger

assert(locateDifference(cast, 5, roles, 4) == 2);

//one runs out, return smaller

assert(locateDifference(cast, 2, roles, 1) == 1);

//elements number cannot be negative

assert(locateDifference(cast, -5, roles, 1) == -1);

//one string size equal to 0

assert(locateDifference(cast, 0, roles, 4) == 0);

//both string size equal to 0

assert(locateDifference(cast, 0, roles, 0) == 0);

cerr << "PASSED: locateDifference" << endl;

//duplicates test cases

string dups1[6] = {"lana","malory", "malory", "malory", "sterling","sterling"};

// //normal case

assert(eliminateDups(dups1, 6) == 3);

assert(dups1[0] == "lana" && dups1[1] == "malory" && dups1[2] == "sterling");

//no duplicates anywhere

string dups2[3] = {"lana", "clarice", "jenny"};

assert(eliminateDups(dups2, 3) == 3);

//element size is zero

assert(eliminateDups(dups2, 0) == 0);

//negative number of elements

assert(eliminateDups(dups2, -5) == -1);

//string e[4] = {"cat", "bird", "dog", "dog"};

cerr << "PASSED: eliminateDups" << endl;

//subsequence test cases

string big1[10] = {"malory", "lana", "sterling", "cheryl", "cyril", "sterling"};

string little1[10] = {"lana", "cheryl", "cyril"};

string little2[10] = { "sterling", "lana" };

//true case

assert(subsequence(big1, 6, little1, 3) == true);

//false case

assert(subsequence(big1, 6, little2, 2) == false);

//0 elements big array

assert(subsequence(big1, 0, little2, 3) == false);

//0 elements in small array, empty subsequence

assert(subsequence(big1, 6, little2, 0) == true);

//negative elements

assert(subsequence(big1, -6, little1, 3) == false);

assert(subsequence(big1, 6, little1, -3) == false);

cerr << "PASSED: subsequence" << endl;

string t[5] = { "cheryl", "krieger", "lana", "malory", "pam" };

string bad[3] = {"zack", "anna", "malory"};

string badz[8];

string res[10];

//one string isn't in nondecreasing order

assert(makeMerger(t, 5, bad, 3, badz, 8) == -1);

assert(makeMerger(t, 5, t, 5, res, 10) == 10);

for(int i = 0; i < 10; i+=2) {

assert(res[i] == res[i + 1] && res[i] == t[i/2]);

}

//normal makeMerger case

assert(makeMerger(t, 2, t, 3, res, 10) == 5);

cerr << "PASSED: makeMerger" << endl;

//divide test cases

//normal test cases

string ppl6[6] = { "krieger", "cheryl", "pam", "cyril", "sterling", "lana" };

assert(divide(ppl6, 6, "malory") == 4); // returns 4

string ppl7[4] = { "malory", "lana", "sterling", "cheryl" };

assert(divide(ppl7, 4, "lana") == 1); // returns 1

// all elements less < divider

string ppl8[4] = {"abigal", "anna", "cheryl", "asdf"};

assert(divide(ppl8, 4, "zack")== 4);

//all elements > divider, return 0

string ppl9[4] = {"helen", "betsy", "clarice", "denis"};

assert(divide(ppl9, 4, "anna") == 0);

//no elements; this should return 0

assert(divide(ppl9, 0, "elyse") == 0);

//negative number of elements

assert(divide(ppl9, -4, "elyse") == -1);

cerr << "PASSED: divide" << endl;