Madeleine Murphy

004991129

11/6/18

Project 4 Report

1. The most notable obstacles overcame were getting through the subsequence and divide functions. Most of the other functions I was able to figure out either with one or two tries, or with line by line debugging. However, subsequence and divide took me a long time to figure out completely and I had to run many tests to make sure they were correct.

For example, one of the bug fixes I had to make was making sure that subsequence could deal with the test of n2 = n1:

int y = (subsequence(sub1, 4, sub2, 4));

added in case where they have same length:

if (n2 == n1)

    {

        for (int k = 0; k < n1; k++)

        {

            if (a1[k] != a2[k])     //if we have the case where they are same length, every position must match

                return -1;

        }

        return 0;

    }

I also just had to go through line by line often in subsequence many times to make sure every little detail was correct. Another detail that was important was making k <= n1 not just <n1 in the case such as

string sub1[] = {"blah", "wubba", "lubba", "rick", "morty"};

    string sub2[] = {"wubba", "lubba", "rick", "morty"};

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

where it’s in there but needs to go up to 1, not just 0, to see that.

For divide, I had a few notable obstacles I had to overcome. First, it took me a long time and many different attempts to figure out that I first needed to sort the array alphabetically, and then to check for the first position greater than the divider. I also had to go back and forth many times with the placeHolder value and the a[k] and a[i] to make sure the swapping of strings was done correctly. Finally, I did a test where all the strings in the test were less than the divider, and realized an important bug:

assert(divide(stuff5, 5, "ez")==5); //test if n is returned if all strings are less than "ez"

            //return n not negative 1 at bottom!

Added in at the end of divide function

    return n;   //one assert failed, had to remember to return n not -1!

These were just a few of the major issues I encountered or important bug fixes I made. There were many others along the way as well.

1. Assert statements to test data! Used a variety of assert sequences to test data. Additionally, as can be seen in my code, throughout the process I used basic tests similar to those in the spec to do preliminary checks along the way before hashing out every nitty gritty detail. For space sake, I won’t include them here, because they were often similar versions to the assert statements just in a non-assert form, or were the exact tests provided in the spec (maybe with a little variation)

Assert tests:

To test lookup & positionofMax function:

string h[7] = { "birdperson", "lois", "morty", "chris", "", "rick", "morty" };

  assert(lookup(h, 7, "rick") == 5);

  assert(lookup(h, 7, "morty") == 2);

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

  assert(positionOfMax(h, 7) == 5);

To test append to all & then rotating left:

string g[4] = { "rick", "morty", "summer", "stewie" };

  assert(appendToAll(g, 4, "!!") == 4 && g[0] == "rick!!" && g[3] == "stewie!!");

  assert(rotateLeft(g, 4, 1) == 1 && g[1] == "summer!!" && g[3] == "morty!!");

string people[6] = { "a", "b", "c", "b", "d", "b" };

    int j = appendToAll(people, 6, "!!!");  // returns 6

    assert(j==6);

    assert(people[0]=="a!!!");

    assert(people[1]=="b!!!");

    assert(people[5]=="b!!!");

Rotate left tests (check that they are in proper places and return right value):

string humanz[5] = { "gloria", "sam", "bart", "lisa", "stefano" };

    int m = rotateLeft(humanz, 5, 1);

    assert(m==1);

    assert(humanz[0]=="gloria");

    assert(humanz[1]=="bart");

    assert(humanz[2]=="lisa");

    assert(humanz[3]=="stefano");

    assert(humanz[4]=="sam");

Test flip (make sure properly flipped and value returned correctly)

string humans[6] = { "mark", "niko", "", "mads", "gaby", "colin" };

    int q = flip(humans, 4);

    assert(q==4);

    assert(humans[0]=="mads");

    assert(humans[1]=="");

    assert(humans[2]=="niko");

    assert(humans[3]=="mark");

    assert(humans[4]=="gaby");

    assert(humans[5]=="colin");

Differ tests:

string test1[] = {"yuh", "yeet", "yah"};

    string test2[] = {"yuh", "YEET", "yah"};

    assert(differ(test1, 1, test2, 1)==1); //the smaller n should be given: 1, both arrays equal

    assert(differ(test1, 2, test2, 1)==1); //verifies that an irrelevant change in size makes no difference

    assert(differ(test1, 1, test2, 2)==1); //same

    assert(differ(test1, 2, test2, 2)==1); //since a difference has been found, the index 1 where it occurs is returned

    assert(differ(test1, 3, test2, 3)==1); //since a difference is found halfway in, the rest of the array doesn't matter

    assert(lookupAny(test1, 1, test2, 1)==0); //first match, 0 is always returned

    assert(lookupAny(test1, 3, test2, 3)==0); //same

Count runs tests:

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

    assert(countRuns(d, 5) == 2);

    string p[9] = {

            "xavier", "betty", "john", "john", "ed", "ed", "ed", "john", "john"};

    assert(countRuns(p, 9) == 5);

    string w[9] = {

        "betty", "betty", "xavier", "john", "ed", "mary", "ed", "ed", "john"};

    assert(countRuns(w, 9) == 7);

Subsequence tests:

string sub1[] = {"blah", "wubba", "lubba", "rick", "morty"};

    string sub2[] = {"wubba", "lubba", "rick", "morty"};

    assert(subsequence(sub1, 5, sub2, 4)==1);

    assert(subsequence(sub1, 4, sub2, 4)==-1); //sub2 is found in sub1, but we can't see the part where they completely coincide

    assert(subsequence(sub1, 4, sub2, 3)==1); //see if sub2 can shift correctly and be found in sub1 at index 1

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

    //checking cases where they have same length

    string sub3[] = {"blah", "wubba", "lubba", "rick"};

    string sub4[] = {"wubba", "lubba", "rick", "morty"};

    string sub5[] = {"blah", "wubba", "lubba", "rick"};

    assert(subsequence(sub3, 4, sub4, 4)==-1);

    assert(subsequence(sub3, 4, sub5, 4)==0);

Divide tests:

    string stuffAns[] = {"art", "bat", "cat", "dolphins", "elephant"};

    string stuff1[] = {"art", "bat", "cat", "dolphins", "elephant"};

    string stuff2[] = {"art", "bat", "cat", "dolphins", "elephant"};

    string stuff3[] = {"art", "bat", "cat", "dolphins", "elephant"};

    string stuff4[] = {"art", "bat", "cat", "dolphins", "elephant"};

    string stuff5[] = {"art", "bat", "cat", "dolphins", "elephant"};

    assert(divide(stuff1, 5, "cat")==2);

    assert(divide(stuff2, 5, "art")==0);

    assert(divide(stuff3, 5, "az")==1);

    assert(divide(stuff4, 5, "ele")==4);

    assert(divide(stuff5, 5, "ez")==5); //test if n is returned if all strings are less than "ez"