Jason Tran

CS31 Project 4

11/2/22

A)

A notable obstacle I overcame was trying to find a way to only use one array, for the three functions, flip, rotateLeft, and split. To solve this problem, I basically just created a temp string and changed it to the string value that I currently needed.

Another problem I had was figuring out how to solve the split function. This was tricky because it wanted the splitter to be in between string values that were less than and greater than. This was hard since the array is probably not going to be sorted, so it will be hard to figure out whether each element would be on the right or left of split. To overcome this challenge, I basically had to figure out a way to sort the array first. The simplest way I learned how to was to just keep switching an element to the right if it is greater than the one next to it. To simplify the sorting function, I tested it with integers first since you can tell if they are sorted better than strings.

Another obstacle that impacted most of my functions was considering which cases were erroneous (will return –1) and which cases were actually allowed, such as if flip was 0.Since there were 10 functions, a lot of edge cases had to be identified, which made this project take longer than expected.

B)

**//testing appendToAll**

string a[8] = { "a","b","c","d","e","f","g","h" };

* assert(appendToAll(a, 8, "zzz") == 8 && a[0] == "azzz" && a[7] == "hzzz"); **/**/test if the function appends correctly to every element
* assert(appendToAll(a, -5, "zzz") == -1);//check if invalid n size
* assert(appendToAll(a, 3, "zzz") == 3 && a[0] == "azzzzzz" && a[3] == "dzzz");//only modifies the first n elements
* assert(appendToAll(a, 0, "zzz") == 0);//0 elements array

**//testing lookup**

string b[8] = { "a","b","c","d","e","f","g","h" };

* assert(lookup(b, 8, "h") == 7); **//**test if functions lookups correctly
* assert(lookup(b, 5, "h") == -1); //if array is shrunk, element should not be found
* assert(lookup(b, 8, "z") == -1);// no target in array
* assert(lookup(b, 0, "a") == -1); //0 element array

**//testing positionOfMax**

string c[7] = { "a", "d" ,"c", "z", "b", "f", "z" };

* assert(positionOfMax(c, 6) == 3); //checks if finds max position right
* assert(positionOfMax(c, 2) == 1); //smaller array should have diffrent max
* assert(positionOfMax(c, 0) == -1); //0 element array is no max
* assert(positionOfMax(c, 7) == 3); //chooses smaller position of max

**//testing roateLeft**

string d1[5] = { "dad", "mom", "sister", "brother", "me" };

* assert(rotateLeft(d1, 5, 1) == 1 && d1[1] == "sister" && d1[2] == "brother" && d1[4] == "mom"); //moves pos value to last and rotates left correctly.
* string d2[5] = { "dad", "mom", "sister", "brother", "me" };
* assert(rotateLeft(d2, 5, 5) == -1); // choosing invalid position of array
* assert(rotateLeft(d2, 5, 4) == 4 && d2[4] == "me" && d2[1] == "mom"); // nothing should change as you are rotating the last position

**//testing countRuns**

string e[9] = {

"tony", "boris", "rishi", "rishi", "gordon", "gordon", "gordon", "rishi", "rishi"

};

* assert(countRuns(e, 9) == 5); //testing of countRuns is correct
* assert(countRuns(e, -3) == -1); //negative size should return -1
* assert(countRuns(e, 1) == 1); //shrinking array size should change # of runs
* assert(countRuns(e, 0) == 0); //n =0

**//testing flip**

string f1[4] = { "a","b","c","d"};

* assert(flip(f1, -1) == -1 && f1[0] == "a" && f1[3] =="d"); //invalid flip
* assert(flip(f1, 0) == 0 && f1[0] == "a" && f1[3] == "d"); //n=0 everything stays the same
* assert(flip(f1, 4) == 4 && f1[0] == "d" && f1[1] == "c" && f1[2] == "b" && f1[3] == "a"); //n is an even flip, and test if flip actually reverses

string f2[5] = { "a","b","c","d","e" };

* assert(flip(f2, 5) == 5 && f2[0] == "e" && f2[1] == "d" && f2[2] == "c" && f2[3] == "b"); // n is an odd flip and actually reverses

**//testing differ**

string g[5] = { "a","b","c","d","e" };

string g1[5] = { "a","b","c","z","e" };

* assert(differ(g, 5, g1, -1) == -1); //invalid n value because negative
* assert(differ(g, 5, g1, 0) == 0); //should be the first position
* assert(differ(g, 5, g1, 5) == 3); //check if finds correct position
* assert(differ(g, 3, g1, 3) == 3); //everything equal and same size

**//testing subsequence**

string h[5] = { "a","b","c","d","e" };

string h1[2] = { "a","b" };

* assert(subsequence(h, 2, h1, 4) == -1); //if the target is bigger than the array
* assert(subsequence(h, 5, h1, 2) == 0); //check if runs correctly
* assert(subsequence(h, 5, h1, 0) == 0); //n2 is 0
* assert(subsequence(h, 0, h1, 0) == 0); //both arrays are empty

string h2[5] = { "b","a","c","a","b"};

* assert(subsequence(h2, 5, h1, 2) == 3); //finds the correct start location of sequence.

**//testing lookupAny**

string i[5] = { "d","e","b","a","e" };

string i1[2] = { "a","b" };

* assert(lookupAny(i, 5, i1, -3) == -1); // n2 is negative
* assert(lookupAny(i, 5, i1, 2) == 2); //finds the smallest position
* assert(lookupAny(i, 3, i1, 1) == -1); //element is not in the array
* assert(lookupAny(i, 5, i1, 0) == -1); //n2 is 0
* assert(lookupAny(i, 0, i1, 1) == -1); //n1 is 0, so no element found

**//testing split**

string j[5] = { "p","c","b","f","r" };

* assert(split(j, 5, "m") == 3 && j[3] == "p"); //splits without splitter in array
* assert(split(j, 5, "z") == 5 && j[4] == "r"); //nothing is greater than the splitter
* assert(split(j, 0, "a") == 0); //array size 0
* assert(split(j, 5, "p") == 3 && j[0] == "b" && j[3]=="p"); //splitter is in the array
* assert(split(j, 5, "a") == 0 && j[0] == "b"); //when every element is bigger than the splitter