Rahul Reddy

CS 31

Professor Smallberg

Report for Project 4

1. A brief description of notable obstacles you overcame.

When looking back at this project, I believe that I overcame a few major obstacles. The first one of these was actually understanding what each function did. I think that once I understood what each function did and was able to write/draw it out on paper, it was much easier to come up with an algorithm/solution to solve it. For example, it was sometimes tough to figure out how exactly the “divide” and “subsequence” functions worked along with how the parameters were important to the function.

I think the other major hurdle was making sure that some of my functions (that rearranged array elements in some way, shape, or form) were printing out the correct order when I would test it out on my own. For these functions, it was easy to return the right value, but making sure the function did what it was supposed to was another challenge on its own. This was because you had to actually test it out.

1. A list of the test data that could be used to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.

appendToAll TESTS:

string f[3] = { "fought", "fight", "fire" };

assert(appendToAll(f, 3, "!") == 3 && f[0] == "fought!" && f[2] == "fire!"); //checks specific cases to see if the function returns the right appended strings

string words[7] = { "hi", "bye", "hello", "goodbye", "greetings", "hiya", "hola" };

assert(appendToAll(words, 2, "!!!") == 2); //checks to see if the appendToAll function returns the right value as expected

assert(appendToAll(words, -3, "!") == -1); //checks to see if the appendToAll function returns -1 when n is negative

assert(appendToAll(words, 0, "!") == 0); //checks to see if the appendToAll function returns 0 with a 0 value

positionOfMax TESTS:

assert(positionOfMax(words, 7) == 6); //checks to see if the positionOfMax function works and returns correct position

assert(positionOfMax(words, 4) == 0); //checks to see if the positionOfMax function works and returns correct position when only part of the array is checked

assert(positionOfMax(words, -4) == -1); //checks to see if the positionOfMax function returns -1 when n is negative

assert(positionOfMax(words, 0) == -1); //checks to see if the positionOfMax function returns -1 when n is 0

lookup TESTS:

string names[7] = {"jose", "john", "joe", "james", "jorge", "bob", "maddie" };

assert(lookup(names, 7, "john") == 1); //checks to see if the lookup function does what it is supposed to

assert(lookup(names, 7, "jose") == 0); //checks to see if the lookup function does what it is supposed to

assert(lookup(names, 2, "ed") == -1); //checks to see if the lookup function returns -1 when no such string is in the array

assert(lookup(names, 2, "maddie") == -1); //checks to see if the lookup function returns -1 when the string is in the array butout of bounds in terms of the parameter

assert(lookup(names, -3, "john") == -1); //checks to see if the lookup function returns -1 n is negative

assert(lookup(names, 0, "john") == -1); //checks to see if the lookup function returns -1 when n is 0

rotateLeft TESTS:

string colleges[4] = {"usc", "ucla", "stanford", "caltech"};

assert(rotateLeft(colleges, 4, 1) == 1); //checks if the rotateLeft function returns pos

assert(rotateLeft(colleges, 3, 1) == 1); //checks if the rotateLeft function returns pos when n is less than the total number of elements

assert(rotateLeft(colleges, 4, 1) == 1); //checks if the rotateLeft function returns pos

assert(rotateLeft(colleges, 4, 1) == 1 && colleges[1] == "ucla" && colleges[3] == "stanford"); //checks specific elements within the array after the rotate left function

assert(rotateLeft(colleges, 4, 4) == -1); //checks if -1 is returned when pos >= n

assert(rotateLeft(colleges, 0, 4) == -1); //checks -1 is returned when n=0

countRuns TESTS:

string names2[4] = { "reed", "bob", "tony", "harold"};

assert(countRuns(names2, 4) == 4); // checks if the countRuns function works as specified

assert(countRuns(names2, 3) == 3); //checks if the countRuns function works as specified when you make the int parameter less elements

assert(countRuns(names2, 0) == 0); //checks if the countRuns function returns 0 when n is 0

assert(countRuns(names2, -3) == -1); //checks if the countRuns function return -1 when n is negative

string pokerHands[5] = {"highCard", "pair", "pair", "twoPair", "twoPair"};

assert(countRuns(pokerHands, 5) == 3); //checks if countRuns function works when there are a few repeated strings

assert(countRuns(pokerHands, 0) == 0); //checks if countRuns returns 0 when n is 0

flip TESTS:

string states[3] = { "az", "ca", "dc" };

//checks to see if array is reversed and elements are in their newly assigned places

assert(flip(states, 3) == 3 && states[0] == "dc" && states[2] == "az");

string folks[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

assert(flip(folks, 4) == 4 && folks[0] == "susan" && folks[2] == "mike"); //checks to see if flip returns 4 since the array now has "susan" at 0 and "mike" at 2

assert(flip(folks, -4) == -1); //checks to see if flip returns -1 since n is -4

assert(flip(folks, 0) == 0); //checks if flip returns 0 when n is 0

differ TESTS:

string name[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

string group[5] = { "donald", "mike", "jamie", "", "susan" };

assert(differ(name, 6, group, 5) == 2); //checks if the differ function returns 2 like it should

assert(differ(name, 2, group, 1) == 1); //checks if the differ function returns 1 like it should

assert(differ(name, 0, group, 4) == 0); //checks if the differ function returns 0 when one of either n1 or n2 is 0

assert(differ(name, 0, group, -3) == -1); //checks if the differ function returns -1 when one of either n1 or n2 is negative

assert(differ(name, 0, group, 0) == 0); //checks if the differ function returns 0 when BOTH n1 or n2 are 0

subsequence TESTS:

string v[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey" };

string e[4] = { "joe", "susan", "", "kamala" };

assert(subsequence(v, 7, e, 4) == 2); //makes sure that subsequence works as specified

assert(subsequence(v, 7, e, 0) == 0); //makes sure that subsequence returns 0 when n2 is 0

assert(subsequence(v, 0, e, 0) == 0); //makes sure that subsequence returns 0 when both n1 and n2 are 0

assert(subsequence(v, -1, e, 3) == -1); //makes sure that subsequence returns -1 when either n1 or n2 is negative

assert(subsequence(v, 2, e, 3) == -1); //makes sure that subsequence returns -1 when n1 < n2 is negative

lookupAny TESTS:

string nam[6] = { "kamala", "mark", "sara", "martha", "donald", "lindsey" };

string set1[4] = { "jamie", "donald", "martha", "mark" };

assert(lookupAny(nam, 6, set1, 4) == 1); //checks if lookupAny returns 1 since a1 has "mark"

assert(lookupAny(nam, 6, set1, 0) == -1); //checks if lookupAny returns -1 when n2 is 0

assert(lookupAny(nam, 0, set1, 6) == -1); //checks if lookupAny returns -1 when n1 is 0

string set2[2] = { "susan", "joe" };

assert(lookupAny(nam, 6, set2, 2) == -1); //checks if lookupAny returns -1 since a1 has none

string h[7] = { "martha", "mark", "joe", "susan", "", "kamala", "lindsey"};

string n[3] = { "lindsey", "joe", "mike" };

assert(lookupAny(h, 7, n, 3) == 2); //checks to see if lookup returns 2 in this case

divide TESTS:

string candidate[6] = { "jamie", "lindsey", "mark", "susan", "joe", "donald" };

assert(divide(candidate, 6, "kamala") == 3); //checks to see if divide returns 3

string candidate2[4] = { "mark", "sara", "lindsey", "mike" };

assert(divide(candidate2, 4, "mike") == 2); //checks to see if divide returns 3

assert(divide(candidate2, -3, "mike") == -1); //checks to see if divide returns -1 when a negative n number is entered

assert(divide(candidate2, 1, "zaza") == 1); //checks to see if divide returns n when the entire array is smaller than the divider

assert(divide(candidate2, 0, "zaza") == 0); //checks to see if 0 is returned n = 0