Joshua Liu

105136031

November 4, 2018

CS 31

Description of Obstacles I Overcame

While working on the program, I found it hard to figure out the correct ways to look through the arrays, ensuring that I was not accessing any invalid positions that might cause the program to fail. Developing a way to create the function was a little difficult for me, especially the subsequence and lookupAny function. To overcome this obstacle, I used a pencil and paper to trace out my program with various data sets to see if the program was evaluating the correct spots in the arrays and returning the correct values.

Data to Test Program

Below is main code used to test my program. After each assert statement is the reason for the test. All tests cases run correctly.

//appendToAll

string a[5] = { "banana", "apple", "", "chicken", "tomato" };

assert(appendToAll(a, 5, "!!!") == 5 && a[2] == "!!!" && a[4] == "tomato!!!"); // append "!!!" to all the strings in the array

assert(appendToAll(a, 3, "=") == 3 && a[1] == "apple!!!=" && a[3] == "chicken!!!"); // append "=" to the first 3 elements in the array

assert(appendToAll(a, 0, "@") == 0 && a[0] == "banana!!!="); // n is zero, return 0

assert(appendToAll(a, -3, "@") == -1 && a[0] == "banana!!!="); // n is negative, return -1

//lookup

string b[5] = { "banana", "apple", "", "chicken", "apple" };

assert(lookup(b, 2, "apple") == 1); // look for the first apple(element 2)

assert(lookup(b, -4, "apple") == -1); // negative n, return -1

assert(lookup(b, 5, "bAnana") == -1); // case matters, return -1

assert(lookup(b, 0, "banana") == -1); // n is zero, nothing in array so return -1

//positionOfMax

assert(positionOfMax(b, 5) == 3); // "chicken" is largest string, return 3

assert(positionOfMax(b, -5) == -1); // negative n, return -1

assert(positionOfMax(b, 0) == -1); // no elements, return -1

string c[6] = { "banana", "apple", "", "chicken", "apple", "chicken" };

assert(positionOfMax(c, 6) == 3); // return 3 since that is the smallest position of the largest string

//rotateLeft

assert(rotateLeft(c, 6, 2) == 2 && c[1] == "apple" && c[2] == "chicken" && c[5] == ""); // rotate string and return the original postion

string d[6] = { "banana", "apple", "", "chicken", "apple", "chicken" };

assert(rotateLeft(d, 2, 2) == -1); // return -1 since position is invalid if equal to n

assert(rotateLeft(d, -2, -2) == -1); // return -1 since position and n are negative

assert(rotateLeft(d, 6, 0) == 0 && d[0] == "apple" && d[5] == "banana"); // return position 0, rotate

string e[6] = { "banana", "apple", "", "chicken", "apple", "chicken" };

assert(rotateLeft(e, 6, 5) == 5 && e[0] == "banana" && e[5] == "chicken"); // rotate last, no change since already in last position

//countRuns

string f[6] = { "banana", "apple", "banana", "banana", "chicken", "chicken" };

assert(countRuns(f, 6) == 4); // count runs for regular string

assert(countRuns(f, 1) == 1); // only 1 element

assert(countRuns(f, -3) == -1); // negative n, return -1

assert(countRuns(f, 0) == 0); // no elements, so return 0 runs

//flip

string g[6] = { "banana", "apple", "chicken", "egg", "chicken", "grape" };

assert(flip(g, 6) == 6 && g[0] == "grape" && g[2] == "egg" && g[5] == "banana"); // flip and return n

string h[6] = { "banana", "apple", "chicken", "egg", "chicken", "grape" };

assert(flip(h, 3) == 3 && h[0] == "chicken" && h[2] == "banana" && h[5] == "grape"); // only flip first 3 elements and return n

string i[6] = { "banana", "apple", "chicken", "egg", "chicken", "grape" };

assert(flip(i, 0) == 0 && i[0] == "banana" && i[5] == "grape"); // flip 0 elements, return 0

assert(flip(i, -2) == -1); // negative elements, return -1

//differ

string j[6] = { "banana", "apple", "chicken", "egg", "chicken", "grape" };

string k[6] = { "banana", "apple", "chicken", "elephant", "chicken", "grape" };

assert(differ(j, 6, k, 6) == 3); // regular data, return 3 for position of different strings

assert(differ(j, 2, k, 2) == 2); // both same number of elements, return 2

assert(differ(j, 2, k, 3) == 2); // return 2 since string j ends

assert(differ(j, -2, k, 3) == -1); // return -1, negative element

assert(differ(j, 0, k, 3) == 0); // return 0 since one has zero elements

//subsequence

string l[6] = { "banana", "egg", "banana", "egg", "chicken", "grape" };

string m[2] = { "egg", "chicken" };

assert(subsequence(l, 6, m, 2) == 3); // array of string with subsequence at position 3

assert(subsequence(l, 6, m, 1) == 1); // 1 element subsequence

assert(subsequence(l, 0, m, 0) == 0); // 0 element subsequence and sequence, return 0

assert(subsequence(l, 6, m, -1) == -1); // negative subsequence, return -1

assert(subsequence(l, 1, m, 2) == -1); // subsequence greater than sequence, return -1

//lookupAny

string n[6] = { "banana", "egg", "banana", "egg", "chicken", "grape" };

string o[2] = { "chicken", "egg" };

assert(lookupAny(n, 6, o, 2) == 1); // look for "chicken" or "egg" in array n, return 1 since "egg" comes first

assert(lookupAny(n, 1, o, 2) == -1); // element n1 is less than element n2, return -1

assert(lookupAny(n, 1, o, 0) == -1); // 0 elements, return -1 since not in array n

string p[2] = { "pineapple", "strawberry" };

assert(lookupAny(n, 6, p, 2) == -1); // no match, return -1

//divide

string q[6] = { "banana", "egg", "banana", "egg", "chicken", "grape" };

assert(divide(q, 6, "dog") == 3 && q[2] == "chicken" && q[3] == "egg" && q[5] == "grape"); // "dog" is betwwen chicken and egg, return position 3

string r[6] = { "banana", "egg", "banana", "egg", "chicken", "grape" };

assert(divide(r, 6, "chicken") == 2 && r[2] == "chicken" && r[3] == "egg" && r[5] == "grape"); // "chicken" is after banana and before egg, return position 2, which is chicken

string s[6] = { "banana", "egg", "banana", "egg", "chicken", "grape" };

assert(divide(s, -3, "chicken") == -1); // negative n, return -1

string t[6] = { "banana", "egg", "banana", "egg", "chicken", "grape" };

assert(divide(t, 0, "chicken") == 0); // 0 elements means

string u[6] = { "dog", "egg", "star", "egg", "flag", "grape" };

assert(divide(u, 6, "chicken") == 0); // all elements greater than divider

string v[6] = { "dog", "egg", "star", "egg", "flag", "grape" };

assert(divide(v, 6, "table") == 6); // all elements less than divider

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