Urvi Shah

Project 4 Report

1. One obstacle that I overcame was figuring out some of the properties of arrays. I learned how to use arrays in Java, so I had to be aware of the differences between C++ and Java in relation to arrays. I couldn’t find the size of an array, so I had to use the value that was given in the function. Another obstacle that I overcame was determining if my function was actually working. Sometimes it returned the right value but wasn’t doing the right thing within the function, so I printed out the array to test it.
2. Test Data

string a[8] = {“apple”, “pear”, “”, “peach”, “cherry”, “tomato”, “plum”, “lemon”};

assert(appendToAll(a, 8, "!") == 8 && a[0] =="apple!" && a[2] =="!" && a[7] == “lemon!”);

// This checks that the function runs normally, changing the first, last, and middle elements and // returning the correct number.

assert(appendToAll(a, -40, "!") == -1);

// This checks that the function returns -1 if the n value is less than 0.

assert(appendToAll(a, 0, "?") == 0 && a[0] =="apple!" && a[2] =="!" && a[7] == “lemon!”);

// This checks that the array is left unchanged and the return value is 0 if the n value is 0.

string b[8] = {“apple”, “pear”, “”, “peach”, “cherry”, “tomato”, “plum”, “lemon”};

assert(lookup(b, 8, "peach") == 3 && lookup(b, 8, "lemon") == 7 && lookup(b, 8, "") == 2);

// This checks that the function can find a string if it exists in the array.

assert(lookup(b, 0, "apple") == -1);

// This checks that -1 is returned if the n value is 0.

assert(lookup(b, 8, "gordon") == -1);

// This checks that -1 is returned if the string is not found.

assert(lookup(b, -4, "tomato") == -1);

// This checks that -1 is returned if the n value is less than 0.

assert(positionOfMax(b, 8) == 5 && positionOfMax(b, 5) == 1);

// This checks that the function can find the maximum string.

assert(positionOfMax(b, -5) == -1);

// This checks that -1 is returned if the n value is less than 0.

assert(positionOfMax(b, 0) == -1);

// This checks that -1 is returned if the n value is less than 0.

assert(rotateLeft(b, 8, 1) == 1 && b[1] == "" && b[7] == "pear");

//This checks that the function can correctly rotate left.

assert(rotateLeft(b, 5, 3) == 3 && b[3] == "tomato" && b[4] == "cherry");

// This checks that the function can rotate left even if the n value is smaller than the array size.

assert(rotateLeft(b, 0, 3) == 3 && b[3] == "tomato" && b[4] == "cherry");

//This checks that the function does nothing and returns position if n is less than pos.

assert(rotateLeft(b, 0, -15) == -1 && rotateLeft(b, -20, 3) == -1 && b[3] == "tomato" && b[4] == "cherry");

// This checks that the function returns -1 if n or pos is less than 0.

string c[8] = {“apple”, “apple”, “pear”, “lemon”, “lemon”, “peach”, “”, “peach”};

assert(countRuns(c, 8) == 6);

//This check that the function can reliably count how many times the strings appear.

assert(countRuns(c, 0) == 0);

//This checks that the function returns 0 if the n value is 0 because there are no values.

assert(countRuns(c, -30) == -1);

// This checks that the function returns -1 if the n value is less than 0.

assert(flip(c, 3) ==3 && c[0] =="pear" && c[2] =="apple" && c[1] ==“apple” && c[3] ==“lemon”);

//This checks that the function can flip an odd number that is less than the overall array size.

string d[4] = {“apple”, “pear”, “peach”, “cherry”};

assert(flip(d, 4) == 4 && d[1] =="peach" && d[3] =="apple”);

//This checks that the function can flip an even number of values and returns the position.

assert(flip(d, -23) == -1 && d[1] =="peach" && d[3] =="apple”);

//This checks that the function returns -1 if n is less than 0 and leaves the array unchanged.

assert(flip(d, 0) == 0 && d[1] =="peach" && d[3] =="apple”);

//This checks that the function returns 0 but leaves the array unchanged if n equals 0.

string e[3] = {“apple”, “pear”, “peach”};

string f[3] = {“apple”, “peach”, “pear”};

string g[4] = {“apple”, “pear”, “peach”, “cherry”};

assert(differ(e, 3, f, 3) == 1 && differ(f, 3, e, 3) == 1);

// This checks that the function can find the place where they are not equal and can find it // regardless of order.

assert(differ(e, 3, g, 3) == 3 && differ(g, 3, e, 3) == 3);

//This checks that the function returns the size of the smaller array if the array runs out.

assert(differ(e, -40, g, 3) == -1 && differ(g, 3, e, -2) == -1);

//This checks that the function returns -1 if any of the n values are less than 0.

assert(subsequence(g, 4, e, 3) == 0);

//This checks that the function returns the position of the string that is within

assert(subsequence(e, 3, g, 4) == -1 && subsequence(g, 4, f, 3) == -1);

// This checks that the function returns -1 if the second array is larger than the first array or if //the second array is not found within the first array.

assert(subsequence(e, -20, g, 4) == -1 && subsequence(g, 4, f, -4) == -1);

//This checks that the function returns -1 if either n values are less than 0.

assert(subsequence(g, 4, e, 0) == 0);

//This checks that the function returns 0 if the size of the second array is 0.

string h[3] = {“apple”, “peach”, “pear”};

string i[5] = {“lemon”, “tomato”, “plum”, “pear”, “orange”};

string j[3] = {“orange”, “lemon”, “peach”};

string k[2] = {“chocolate”, “vanilla”};

assert(lookupAny(i, 5, h, 3) == 3);

// This checks that the function can find a string that is in the first array.

assert(lookupAny(i, 5, j, 3) == 0);

// This checks that the function returns the earliest instance in the first array of something in // the second array being equal to it.

assert(lookupAny(i, 5, k, 2) == -1);

// This checks that the function returns -1 if no elements in the arrays equal each other.

assert(lookupAny(i, -34, j, 3) == -1 && lookupAny(i, 5, j, -20) == -1);

//This checks that the function returns -1 if either n value is less than 0.

assert(separate(i, 5, "pro") == 4);

//This checks that the function can separate the array and return the correct value.

assert(separate(i, 5, "pear") == 2);

//This checks that the function returns the correct value when a string in the array is equal to //the separator.

assert(separate(i, 5, "zebra") == 5);

// This checks that the function returns n if the string is greater than all the strings in the array.

assert(separate(i, 5, "apple") == 0);

// This checks that the function returns 0 if the string is less than all the strings in the array.

assert(separate(i, 0, "apple") == 0);

// This checks that the function returns 0 if the n value is 0.

assert(separate(i, -3, "apple") == -1);

//This checks that the function returns -1 if the n value is less than 0.