Some notable obstacles I overcame include the divide function. At first, I really struggled to see how it would work, but I think I ended up just overcomplicating the whole thing. I also struggled with figuring out which values to return.

**A list of test data include:**

reduplicate (string a[], int n)

* a[1] = {“”}, n = 1, test empty string
* a[1] = {“b”}, n = 1, not empty string
* a[1] = {“b”}, n = 0, zero interesting elements
* a[2] = {“a”, “b”}, n = 2, more than one element in array
* a[1] = {“b”}, n = -1, negative n value

locate (const string a[], int n, string target)

* a[1] = {“”}, n = 1, target = “b”, test empty string
* a[1] = {“b”}, n = 1, target = “b”, not empty string and target is in array
* a[1] = {“a”}, n = 1, target = “b”, not empty string and target is not in array
* a[1] = {“b”}, n = 0, target = “b”, zero interesting elements
* a[2] = {“a”, “b”}, n = 2, target = “b”, more than one element in array and target in array
* a[2] = {“a”, “c”}, n = 2, target = “b”, more than one element in array and target not in array
* a[1] = {“b”}, n = -1, target = “b”, negative n value
* a[2] = {“a”, “b”}, n = 2, target = “”, target is empty string and empty string not in array
* a[2] = {“a”, “”}, n = 2, target = “”, target is empty string and empty string is in array

locationOfMax (const string a[], int n)

* a[1] = {“”}, n = 1, test empty string
* a[1] = {“b”}, n = 1, not empty string
* a[1] = {“b”}, n = 0, zero interesting elements
* a[2] = {“a”, “b”}, n = 2, more than one element in array
* a[3] = {“a”, “b”, “c”}, n = 2, max at the end
* a[3] = {“d”, “b”, “c”}, n = 2, max at the beginning
* a[3] = {“a”, “d”, “c”}, n = 2, max in the middle
* a[1] = {“b”}, n = -1, negative n value

circleLeft (string a[], int n, int pos)

* a[1] = {“”}, n = 1, pos = 0, test empty string and valid pos
* a[1] = {“”}, n = 1, pos = 2, pos >= n
* a[6] = {“a”, “b”, “c”, “d”, “e”, “f”}, n = 6, pos = 2, valid position not empty string array
* a[1] = {“a”}, n = 1, target = “b”, not empty string and target is not in array
* a[1] = {“b”}, n = 0, pos = 0, zero interesting elements
* a[1] = {“b”}, n = -1, target = “b”, negative n value

enumerateRuns (const string a[], int n)

* a[1] = {“”}, n = 1, test empty string
* a[1] = {“b”}, n = 1, not empty string
* a[1] = {“b”}, n = 0, zero interesting elements
* a[2] = {“a”, “b”}, n = 2, more than one element in array
* a[1] = {“b”}, n = -1, negative n value
* a[6] = {“a”, “a”, “b”, “c”, “c”, “c”}, n = 6, some identical in a row

flip (string a[], int n)

* a[1] = {“”}, n = 1, test empty string
* a[1] = {“b”}, n = 1, not empty string
* a[1] = {“b”}, n = 0, zero interesting elements
* a[6] = {“a”, “b”, “c”, “”, “ds”, “test”}, n = 6, more than one element in array
* a[1] = {“b”}, n = -1, negative n value

locateDifference (const string a1[], int n1, const string a2[], int n2)

* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “c”}, n2 = 3, identical arrays identical n values
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “c”}, n2 = 2, identical arrays different n values
* a1[3] = {“a”, “b”, “c”}, n1 = 2, a2[3] = {“a”, “b”, “c”}, n2 = 3, identical arrays different n values (n2 is greater)
* a1[3] = {“a”, “b”, “c”}, n1 = 0, a2[3] = {“a”, “b”, “c”}, n2 = 3, n1 is 0
* a1[3] = {“x”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “x”}, n2 = 3, arrays are different
* a1[3] = {“x”, “b”, “c”}, n1 = -1, a2[3] = {“a”, “b”, “x”}, n2 = 3, negative n1
* a1[3] = {“x”, “b”, “c”}, n1 = 1, a2[3] = {“a”, “b”, “x”}, n2 = -3, negative n2

subsequence (const string a1[], int n1, const string a2[], int n2)

* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “c”}, n2 = 3, identical arrays
* a1[3] = {“a”, “b”, “c”}, n1 = 2, a2[3] = {“a”, “b”, “c”}, n2 = 3, identical arrays but n2 > n1
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[2] = {“a”, “c”}, n2 = 2, not a subsequence
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[2] = {“b”, “c”}, n2 = 2, is a subsequence but not right away
* a1[3] = {“a”, “b”, “c”}, n1 = -3, a2[3] = {“a”, “b”, “c”}, n2 = 3, negative n1
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “c”}, n2 = -3, negative n2
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “c”}, n2 = 0, n2 is 0

locateAny(const string a1[], int n1, const string a2[], int n2)

* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “e”, “d”}, n2 = 3, return right away
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“d”, “e”, “f”}, n2 = 3, no valid position
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[2] = {“b”, “c”}, n2 = 2, do not return right away
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[2] = {“d”, “c”}, n2 = 2, when it is not the first element of a2 that matches
* a1[3] = {“a”, “b”, “c”}, n1 = -3, a2[3] = {“a”, “b”, “c”}, n2 = 3, negative n1
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “c”}, n2 = -3, negative n2
* a1[3] = {“a”, “b”, “c”}, n1 = 3, a2[3] = {“a”, “b”, “c”}, n2 = 0, n2 is 0

divide (string a[], int n, string divider)

* a[3] = {“a”, “b”, “z”}, n = 3, divider = “n”, already in order
* a[4] = {“a”, “b”, “n”, “z”}, n = 3, divider = “n”, has divider inside array
* a[3] = {“a”, “b”, “z”}, n = -1, divider = “n”, negative n
* a[3] = {“a”, “b”, “z”}, n = 0, divider = “n”, n is 0
* a[3] = {“z”, “b”, “n”}, n = 3, divider = “n”, not in order