## Project 4 Report

### Notable Obstacles

* positionOfMax(): I was initially confused how to identify if 0 elements in the array should be considered before realizing I could simply check if n was equal to 0 or not and return -1 accordingly
* countRuns(): I was confused why my code was returning a number one less than the actual number of runs. I realized I was counting the number of changes in runs rather than the runs themselves. I fixed this by simply adding 1 to my counter variable
* subsequence(): I began by simply finding the index of where the first entry in each a2 matched to in a1. I then used a for loop to check if a2 was in a1. From there, I didn’t know what to do if a2 wasn’t in a1 but eventually realized that I had to use a break statement so that the outer for loop could be reached and we could move on to checking the next elements in a1

### Test Data

**int appendToAll(string a[], int n, string value) tests:**

* Test for catching “n is negative” error: ( {“hi”, “hi2”, “”},-1, “bye”) –> Should return -1 and array should be unchanged
* Test for checking that function appends to end of array: ( {“hi”, “hi2”, “”},2, “bye”) –> Should return 2 and array should now be {“hibye”, “hi2bye”, “bye”}

**int lookup(const string a[], int n, string target) tests:**

* Check that uppercase and lowercase do not register as a match ({“a”, “b”}, 2, “A”}) → Should return -1
* Function generally works ({“a”, “b”}, 2, “a”}) → Should return 0
* Test for catching “n is negative” error: ({“a”, “b”}, -2, “a”}) –> Should return -1 and array should be unchanged

**int positionOfMax(const string a[], int n) tests:**

* ({ "doug", "kamala", "melania", "usha", "gwen", "donald" }, 6); // returns 3, since usha is latest in alphabetic order
* ({ "doug", "kamala", "melania", "6usha", "gwen", "donald" }, 6); // returns 2, since melania is latest in alphabetic order
* ({ "doug", "kamala", "melania", "usha", "usha", "donald" }, 6); // returns 3, since the first instance of “usha” is at the third index even though there are two Usha’s
* ({ "doug", "kamala", "melania", "usha", "gwen", "donald" }, -1); // returns -1, since n is negative
* ({ "doug", "kamala", "melania", "usha", "gwen", "donald" }, 0); // returns -1, since no elements are considered part of the array (n=0)

**int rotateLeft(string a[], int n, int pos) tests:**

* ({ "kamala", "donald", "jd", "tim", "joe" }, 5, 1); // returns 1 and array now contains: "kamala", "jd", "tim", "joe", "donald"
* ({ "kamala", "donald", "jd", "tim", "joe" },-15, 1); // returns -1 because n is negative
* ({ "kamala", "donald", "jd", "tim", "joe" },5, -1); //returns -1 because pos is negative
* ({ "kamala", "donald", "jd", "tim", "joe" },5, 5); //returns -1 because pos is >= array size

**int countRuns(const string a[], int n) tests:**

* ({"melania", "doug", "gwen", "gwen", "jill", "jill", "jill", "gwen", "gwen"}, -1); //returns -1 because n is negative
* ({"melania", "doug", "gwen", "gwen", "jill", "jill", "jill", "gwen", "gwen"}, 9); //returns 5 because there are five distinct sequences
* ({"melania", "doug", "gwen", "gwen", "jill", "jill", "jill", "gwen", "gwen"}, 0); //returns 0 because there are no consecutive elements
* ({"jill", "jill", "jill"}, 1); //returns 1 because there is only one sequence
* ({}, 0); //returns 0 because there are no consecutive elements

**int flip(string a[], int n) tests:**

* ({ "kamala", "doug", "", "jill", "jd", "donald" }, 4) – returns 4 and array is now {"jill", "" , "doug", "kamala", "jd", "donald"}
* ({ "kamala", "doug", "", "jill", "jd", "donald" }, -3) – returns -1 because n is negative

int differ(const string a1[], int n1, const string a2[], int n2) tests:

If there are the arrays folks and group like this:

string folks[6] = { "kamala", "doug", "", "jill", "jd", "donald" };

string group[5] = { "kamala", "doug", "donald", "", "jd" };

* differ(folks, 6, group, 5); // returns 2
* differ(folks, 2, group, 1); // returns 1
* differ(folks, -2, group, 1); // returns -1 because n1 is negative
* differ(folks, 2, group, -1); // returns -1 because n2 is negative

**int differ(const string a1[], int n1, const string a2[], int n2) tests:**

string folks[6] = { "kamala", "doug", "", "jill", "jd", "donald" };

string group[5] = { "kamala", "doug", "donald", "", "jd" };

differ(folks, 6, group, 5); // returns 2

differ(folks, 2, group, 1); // returns 1

differ(folks, -2, group, 1); // returns -1 because n1 is negative

differ(folks, 2, group, -1); // returns -1 because n2 is negative

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

If there are arrays like this:

string names[10] = { "kamala", "tim", "usha", "gwen", "donald", "jd" };

string names1[10] = { "tim", "usha", "gwen" };

string names2[10] = { "kamala", "gwen" };

**Tests:**

subsequence(names, 6, names1, 3); // returns 1

subsequence(names, 5, names2, 2); // returns -1

subsequence(names, -5, names2, 2); // returns -1

subsequence(names, 5, names2, -2); // returns -1

Subsequence{names, 2, names1, 3); //returns -1 because n1 is not big enough to include a2 in a1

**int lookupAny(const string a1[], int n1, const string a2[], int n2) tests:**

string names[10] = { "kamala", "tim", "usha", "gwen", "donald", "jd" };

string set1[10] = { "donald", "melania", "gwen", "tim" };

int v = lookupAny(names, 6, set1, 4); // returns 1 (a1 has "tim" there)

string set2[10] = { "jill", "joe" };

int w = lookupAny(names, 6, set2, 2); // returns -1 (a1 has none)

**int separate(string a[], int n, string separator) tests:**

string people[6] = { "doug", "kamala", "melania", "usha", "gwen", "donald" };

separate(people, 6, "joe"); // returns 3

// people must now be

// "doug" "gwen" "donald" "kamala" "usha" "melania"

// or "gwen" "donald" "doug" "melania" "kamala" "usha"

// or one of several other orderings.

// All elements < "joe" (i.e., "gwen", "donald", and "doug")

// come before all others

// All elements > "joe" (i.e., "usha", "kamala", and "melania")

// come after all others

string people2[4] = { "jill", "tim", "jd", "joe" };

separate(people2, 4, "joe"); // returns 2

// people2 must now be either

// "jd" "jill" "joe" "tim"

// or "jill" "jd" "joe" "tim"

// All elements < "joe" (i.e., "jd" and "jill") come before all others.

// All elements > "joe" (i.e., "tim") come after all others.

separate(people2, -4, "joe"); //returns -1 because n is negative