**Project 4 Sunshine Array Report**

Notable obstacles: I spent some time trying to implement the three functions that rotateLeft, flip, and split, without using a separate array. For split, I managed to finish it by sorting the array through selection sort. I used cerr to test the results of the arrays afterwards in order. Another obstacle I noticed was that with a large number of functions to implement, I should have written my comments right after completing each function. For the subsequence function, I used a bool isValid to remain true until a continguous function was found and then change it to false then. Afterwards, I used if statement to check isValid and change the value returned only when true. It was harder for me to go over my code several days after just to add comments in to try out my test cases. I had several careless runtime-errors with array out of bounds for some functions that required looping only until n-1, but they could be easily debugged.

int appendToAll(string a[], int n, string value);

For string test[6] = { "carly", "mike", "ted", "bernie", "jeb" };

Test cases:

int shouldReturn5 = appendToAll(test, 5, "...");

Outcome: { “carly…”, “mike…”, “ted…”, “bernie…”, “jeb…”}, returns 5

int shouldReturn1 = appendToAll(test, 1, "...");

Outcome: { “carly…”, "mike", "ted", "bernie", "jeb"}, returns 1

int toReturn = appendToAll(test, 0, "...");

Outcome: Array unchanged, returns 0

int toReturn = appendToAll(test, -1, "...");

Outcome: Array unchanged, returns -1

int lookup(const string a[], int n, string target);

For string folks[6] = { "1", "2", "3", "4", "4", "4" };

For string folks2[6] = { "chris", "ben", "", "ben", "donald", "john" };

Test cases:

int toReturn = lookup(folks, 3, "3");

Outcome: toReturn = 2

int toReturn = lookup(folks, 4, "4");

Outcome: toReturn = 3

int toReturn = lookup(folks, 5, "4");

Outcome: toReturn = 3

int toReturn = lookup(folks, 5, "asdf");

Outcome: toReturn = -1

int toReturn = lookup(folks, -1, "3");

Outcome: toReturn = -1

int toReturn = lookup(folks, 0, "3");

Outcome: toReturn = -1

int positionOfMax(const string a[], int n);

For string folks[6] = { "1", "2", "3", "4", "4", "4" };

For string folks2[6] = { "chris", "ben", "", "ben", "donald", "john" };

For string cand[6] = { "bernie", "hillary", "donald", "marco", "carly", "ben" };

Test cases:

int toReturn = positionOfMax(folks, 4);

Outcome: toReturn = 3

int toReturn = positionOfMax(folks, 6);

Outcome: toReturn = 3

int toReturn = positionOfMax(folks2, 2);

Outcome: toReturn = 0

int toReturn = positionOfMax(folks2, 6);

Outcome: toReturn = 5

int toReturn = positionOfMax(cand, 0);

Outcome: toReturn = -1

int toReturn = positionOfMax(cand, -1);

Outcome: toReturn = -1

int rotateLeft(string a[], int n, int pos);

For string folks[6] = { "1", "2", "3", "4", "4", "4" };

For string folks2[6] = { "chris", "ben", "", "ben", "donald", "john" };

For string cand[6] = { "bernie", "hillary", "donald", "marco", "carly", "ben" };

Test cases:

int toReturn = rotateLeft(folks, 6, 3);

Outcome: toReturn = 3, { “1”, “2”, “3”, “4”, “4”, “4”}

int toReturn = rotateLeft(folks, 6, -1);

Outcome: toReturn = -1, { “1”, “2”, “3”, “4”, “4”, “4”}

int toReturn = rotateLeft(folks, 6, 1);

Outcome: toReturn = 1, { “1”, “3”, “4”, “4”, “4”, “2”}

int toReturn = rotateLeft(cand, 2, 1);

Outcome: toReturn = 1, { “bernie, hillary”}

int toReturn = rotateLeft(cand, 6, 2);

Outcome: toReturn = 1, { “bernie, hillary”, “marco”, “carly”, “ben”, “donald”}

int countRuns(const string a[], int n);

For string d2[9] = {"donald", "donald", "donald", "donald", "donald", "donald", "donald", "donald", "donald" };

For string d[9] = { "ben", "chris", "marco", "marco", "donald", "donald", "donald", "marco", "marco" };

Test cases:

int toReturn = countRuns(d2, 1);

Outcome: toReturn = 1

int toReturn = countRuns(d2, 8);

Outcome: toReturn = 1

int toReturn = countRuns(d2, 0);

Outcome: toReturn = 0

int toReturn = countRuns(d, 8);

Outcome: toReturn = 5

int flip(string a[], int n);

For string d[9] = { "ben", "chris", "marco", "marco", "donald", "donald", "donald", "marco", "marco" };

For string d2[9] = {"donald", "donald", "donald", "donald", "donald", "donald", "donald", "donald", "donald" };

Test cases:

int toReturn = flip(d, 6);

Outcome: toReturn = 6, { “donald”, “donald”, “marco”, “marco”, “chris”, “ben}

int toReturn = flip(d, 1);

Outcome: toReturn = 6, { “ben” }

int toReturn = flip(d2, 3);

Outcome: toReturn = 6, { “donald”, “donald”, “donald”, “donald”, “donald”, “donald”}

int toReturn = flip(d, -1);

Outcome: toReturn = -1

int toReturn = flip(d2, 0);

Outcome: toReturn = 0

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

string folks[6] = { "chris", "marco", "", "ben", "donald", "john" };

string group[5] = { "chris", "marco", "john", "", "carly" };

Test cases:

int toReturn = differ(folks, 6, group, 5);

Outcome: toReturn = 2

int toReturn = differ(folks, 3, group, 5);

Outcome: toReturn = 2

int toReturn = differ(folks, 1, group, 5);

Outcome: toReturn = 1

int toReturn = differ(folks, 0, group, 5);

Outcome: toReturn = 0

int toReturn = differ(folks, 3, group, 0);

Outcome: toReturn = 0

int toReturn = differ(folks, -1, group, 3);

Outcome: toReturn = -1

int toReturn = differ(folks, 3, group, -1);

Outcome: toReturn = -1

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

string names[10] = { "ted", "hillary", "rand", "bernie", "mike", "jeb" };

string names1[10] = { "hillary", "rand", "bernie" };

string names2[10] = { "ted", "bernie" };

Test cases:

int toReturn = subsequence(names, 6, names1, 3);

Outcome: toReturn = 1

int toReturn = subsequence(names, 5, names2, 2);

Outcome: toReturn = -1

int toReturn = subsequence(names, 0, names1, 2);

Outcome: toReturn = -1

int toReturn = subsequence(names, 5, names1, 0);

Outcome: toReturn = 0

int toReturn = subsequence(names, 3, names1, 2);

Outcome: toReturn = 1

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

string names[10] = { "ted", "hillary", "rand", "bernie", "mike", "jeb" };

string set1[10] = { "carly", "mike", "bernie", "hillary" };

string set2[10] = { "ben", "donald" };

Test cases:

int toReturn = lookupAny(names, 6, set1, 4);

Outcome: toReturn = 1

int toReturn = lookupAny(names, 6, set2, 2);

Outcome: toReturn = -1

int toReturn = lookupAny(names, 0, set2, 2);

Outcome: toReturn = -1

int toReturn = lookupAny(names, 3, set2, 0);

Outcome: toReturn = -1

int split(string a[], int n, string splitter);

For string cand[6] = { "bernie", "hillary", "donald", "marco", "carly", "ben" };

string cand2[4] = { "donald", "hillary", "jeb", "ben" };

int toReturn = split(cand, 6, "chris");

Outcome: toReturn = 3

int toReturn = split(cand2, 4, "donald");

Outcome: toReturn = 1

int toReturn = split(cand, 0, "chris");

Outcome: toReturn = 0

int toReturn = split(cand, -1, "chris");

Outcome: toReturn = -1