### Obstacles:

In functions “rotateLeft”, “rotateRight” and “flip”, it’s a little difficult to figure out what index I should put when doing operations like moving elements one place to left/right and reversing the array without using another array.

### Test Data:

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

string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int j = appendToAll(cast, 6, “!!!");

// general test; should return 6;

string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int j = appendToAll(cast, -2, “!!!");

// n is negative and the function should return -1;

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

string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int j = lookup(cast, 6, “maggie");

// general test; should return 4;

string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int j = lookup(cast, 6, “riCk");

// cannot find a fit

string cast[7] = { "glenn", "carl", "carol", "rick", "maggie", “daryl”, “carl”};

int j = lookup(cast, 7, “carl”);

// more than one fit found; should return the smallest position number, which is 1.

string cast[7] = { "glenn", "carl", "carol", "rick", "maggie", “daryl”, “carl”};

int j = lookup(cast, -1, “carl”);

// bad argument; should return -1;

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

string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int k = positionOfMax(cast, 6);

// general test; should return 3;

string cast[6] = { "glenn", "carl", "carol", "rick", "maggie", "daryl" };

int k = positionOfMax(cast, -2);

// bad argument; should return -1;

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

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int m = rotateLeft(characters, 5, 1);

// general test; should return 1;

// characters should contain: "rosita", "sasha", "glenn", "michonne", “bob"

// NOTE: does not handle correctly; the characters become "rosita", "sasha", "glenn", “michonne”;

// FIX: save the value of a[pos] and assign it to the nth element of a;

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int m = rotateLeft(characters, 5, 2);

// another general test;

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int m = rotateLeft(characters, 5, 6);

// pos > n; bad argument; should return -1;

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int m = rotateLeft(characters, 5, -1);

// pos < 0; bad argument; should return -1;

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int m = rotateLeft(characters, -1, 6);

// n < 0; bad argument; should return -1;

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

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int p = rotateRight(characters, 5, 2);

// should return 2

// characters should contain: "sasha", "rosita", "bob", "glenn", “michonne”

// NOTE: does not handle correctly; the characters become “rosita sasha rosita glenn michonne”;

// FIX: do “for-loop” in reversed order

i.e. “ for (int i=1; i<=pos; i—); a[i] = a[i-1];” —> “for (int i=pos; i>0; i—); a[i] = a[i-1];”

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int p = rotateRight(characters, 5, -1);

// bad argument: pos is negative; should return -1;

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int p = rotateRight(characters, -1, 2);

// bad argument: n is negative; should return -1;

string characters[5] = { "rosita", "bob", "sasha", "glenn", "michonne" };

int p = rotateRight(characters, 5, 5);

// bad argument: pos is out of the range of the array; should return -1;

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

string roles[6] = { "abraham", "tara", "", "daryl", "carol", "tyreese" };

int q = flip(roles, 4);

// general test; should return 4

// roles should contain: "daryl" "" "tara" "abraham" "carol" "tyreese"

// NOTE: does not handle correctly; returns “daryl”, “tara”, “”, “daryl”

/\* FIX:

for (int i=0; i<(n-2)/2; i++)

a[i]=a[n-1-i];

–––> for (int i=0; i<=(n-2)/2; i++)

{

string temp = a[i];

a[i] = a[n-1-i];

a[n-1-i] = temp;

}

\*/

string roles[6] = { "abraham", "tara", "", "daryl", "carol", "tyreese" };

int q = flip(roles, -100);

// bad argument: n is negative; should return -1;

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

string roles[6] = { "abraham", "tara", "", "daryl", "carol", "tyreese" };

string group[5] = { "abraham", "tara", "tyreese", "", "maggie" };

int r = differ(roles, 6, group, 5); // general test; should return 2;

int s = differ(roles, 2, group, 1); // the arrays are equal up to the point where group runs out; should return 1;

string roles[6] = { "abraham", "tara", "", "daryl", "carol", "tyreese" };

string group[5] = { "abraham", "tara", "tyreese", "", "maggie" };

int r = differ(roles, 6, group, -1); // bad argument: n2 < 0;

int s = differ(roles, -1, group, 1); // bad argument: n1 < 0;

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

string names[10] = { "sasha", "rick", "beth", "glenn", "bob", "michonne" };

string names1[10] = { "rick", "beth", "glenn" };

int t = subsequence(names, 6, names1, 3); // general test; returns 1

string names2[10] = { "sasha", "glenn" };

int u = subsequence(names, 5, names2, 2); // a1 does not contain a2 as a continuous subsequence; returns -1

string names3[10] = { “shabi”, “sasha”, “glenn”, “sasha”, “glenn” };

int v = subsequence(names3, 5, names2, 2);

// The subsequence appears more than once in a1; should return the smallest beginning position, which is 1.

string names4[10] = { "sasha", "glenn" };

int w = subsequence(names4, 2, names2, 2);

// The length of the two sequences are equal; should return 0;

int w = subsequence(names4, 1, names2, 2);

// The length of the first sequence is less than that of the second; should return -1;

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

string names[10] = { "sasha", "rick", "beth", "glenn", "bob", "michonne" };

string set1[10] = { "maggie", "bob", "glenn", "rick" };

int v = lookupAny(names, 6, set1, 4); // general test;

string set2[10] = { "daryl", "carol" };

int w = lookupAny(names, 6, set2, 2); // no element of a1 is equal to any element in a2

string set3[10] = { “sasha”, “rick”, “carol”, “carol”, “daryl”, “daryl” }

int x = lookupAny(set3, 6, set2, 2); // a1 contains several elements of a2; should return the smallest position number, which is 2.

int y = (names, 5, set1, 0); // n2 = 0; should return -1;

int z = (names, 0, set1, 4); // n1 = 0; should return -1;

int u = (names, -1, set1, -1); // n1<0 and n2<0; should return -1;

int separate(string a[], int n, string separator);

string cast[6] = { "maggie", "carl", "daryl", "rick", "michonne", "carol" };

int x = separate(cast, 6, “glenn");

// general test;

string cast2[4] = { "carol", "rick", "michonne", "daryl" };

int y = separate(cast2, 4, “daryl");

// general test;

int z = separate(cast, 6, “zephyr”);

// no element is not < separator; should return n;