1. An obstacle I had was on removeDups. At first, I was trying to make use of moveToEnd by if it found a similar string it would move the similar string to the end of the array and shift everything else to the left. But with this approach, I was having problems as it would modify the string, then the counter would skip to the next element when the replaced element was still a similar string to the previous one. Therefore, it would not catch/remove the duplicate properly.

After careful thought, however, I was able to overcome this obstacle by abandoning the moveToEnd approach and using one simple for loop to solve this problem. For every other function, I didn’t face any obstacles.

enumerate function:

string d[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

int i = enumerate(d, 9, "amy"); // returns 4

//To see if the function can properly count how many Amys there are

int j = enumerate(d, 5, "ketanji"); // returns 1

//To see if the function can properly count how many Ketanji there are

//when the array size is restricted

int k = enumerate(d, 9, "brett"); // returns 0

//To see if the function returns 0 when the target string does not exist //in the array

int l = enumerate(d, 0, "amy"); //returns 0

//To see if the function returns 0 when the array size is 0

int m = enumerate(d, -1, "amy"); //returns -1

//To see if the function returns -1 when the array size is negative

findMatch

string d[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

int m = findMatch(d, 9, "ketanji"); // returns 4

//To see if it correctly finds the smallest position number of ketanji

int n = findMatch(d, 9, "amy"); //returns 2

//To see if it correctly finds the smallest position number of amy

int p = findMatch(d, 4, "ketanji"); // returns -1

//To see if it correctly returns -1 when there is no Ketanji in the first //four elements of the array

int q = findMatch(d, -1, "ketanji"); //returns -1

//To see if the function returns -1 when the array size is negative

int r = findMatch(d, 0, "ketanji"); //returns -1

//To see if the function returns -1 when the array size is 0 as it would have no match

findRun

string d[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

int b;

int e;

bool b1 = findRun(d, 9, "amy", b, e); // returns true and

// sets b to 2 and e to 3

//To see if the function correctly finds the earliest occurrence of runs //of amy and correctly changes b to the beginning of the run and e to end //of the run

bool b2 = findRun(d, 9, "neil", b, e); // returns true and

// sets b to 1 and e to 1

//To see if the function correctly finds the earliest occurrence of a //single string run and correctly changes b to the beginning of the run //and e to end of the run where in this case b and e should be the same

bool b3 = findRun(d, 6, "ketanji", b, e); //returns true

// sets b to 4 and e to 5

//To see if the function correctly finds the earliest occurrence and

//returns the correct b and e of a run that cuts off due to the limit of //the array size examined

bool b4 = findRun(d, 9, "samuel", b, e); // returns false and

// leaves b and e unchanged

//To see if the function correctly returns false when there is no such //runs in the array and leaves b and e unchanged

bool b5 = findRun(d, 0, "neil", b, e); //returns false and

// leaves b and e unchanged

//To see if the function correctly returns false when the array examined //is empty and leaves b and e unchanged

findMin

string people[5] = { "john", "sonia", "samuel", "elena", "neil" };

int q = findMin(people, 5); // returns 3, since elena is earliest

// in alphabetic order

//To see if the function correctly returns the right index for elena

string people[5] = { "john", "sonia", "elena", "elena", "neil" };

int q = findMin(people, 5);

//To see if the function correctly returns the smallest position number of such a string that are equal and smallest

string people[5] = { "bob", "sonia", "chris", "elena", "neil" };

int q = findMin(people, 5);

//To see if the function correctly returns the right index for bob as bob is earliest in alphabetic order

int q = findMin(people, 0);

//To see if the function correctly returns -1 if the function should //consider no elements to be part of the array

int q = findMin(people, -1);

//To see if the function correctly returns -1 if the function is passed a

//bad argument where array size is negative

moveToEnd

string people[5] = { "john", "sonia", "samuel", "elena", "neil" };

int r = moveToEnd(people, 4, 1); // returns 1

// people now contains: "john" "samuel" "elena" "sonia" "neil"

// the function knew about only 4 elements

//To see if the function correctly returns the original position of the

//original item that was moved to the end and correctly moved all elements

//after the element at index 1 to the left and moved the value eliminated //into the last position that the array knew about.

int s = moveToEnd(people, 5, 3);

// people now contains: "john" "sonia" "samuel" "neil" "elena"

// the function knew about all 5 elements

//To see if the function correctly returns the original position of the

//original item that was moved to the end and correctly moved all elements

//after the element at index 3 to the left and moved the value eliminated //into the last position

int s = moveToEnd(people, 2, 3);

//To see if the function correctly returns -1 when the position is greater //than the array size

int t = moveToEnd(people, 0, 0);

//To see if the function correctly returns -1 when the array is empty but //position is at index 0 (which cannot exist when array is empty)

int u = moveToEnd(people, -1, 2);

//To see if the function correctly returns -1 when the array size is //negative

int v = moveToEnd(people, 5, -1);

//To see if the function correctly returns -1 when the pos index is //negative

moveToBeginning

string people[5] = { "john", "sonia", "samuel", "elena", "neil" };

int s = moveToBeginning(people, 5, 2);

//To see if the function correctly returns the original position of the

//original item that was moved to the beginning and correctly moved all //elements before the element at index 2 to the right and moved the value //eliminated into the first position of the array

int t = moveToBeginning(people, 5, 0);

//To see when the item that is eliminated is the first element it returns

//0 and doesn’t change the array in any way

int u = moveToBeginning(people, 3, 4);

//To see if the function correctly returns -1 when the position is greater //than the array size

int v = moveToBeginning(people, 0, 0);

//To see if the function correctly returns -1 when the array is empty but //position is at index 0 (which cannot exist when array is empty)

int w = moveToBeginning(people, -1, 3);

//To see if the function correctly returns -1 when the array size is //negative

int x = moveToBeginning(people, 5, -1);

//To see if the function correctly returns -1 when the pos index is //negative

findDifference

string people[5] = { "john", "sonia", "samuel", "elena", "neil" };

string bench[6] = { "john", "sonia", "clarence", "elena", "neil", "samuel" };

int r = findDifference(people, 5, bench, 6); // returns 2

//To see if the function correctly returns the position of the first //corresponding elements of a1 and a2 that are not equal (in this case 2)

int s = findDifference(people, 2, bench, 1); // returns 1

//To see if the function correctly returns whichever value of n1 and n2 is //less than or equal to the other when the arrays are equal up to the //point where one or both runs out,

int t = findDifference(people, 0, bench, 6); //returns 0

//To see if the function correctly returns whichever value of n1 and n2 is //less than or equal to the other when one of the array size is 0

int u = findDifference(people, 5, bench, 0); //returns 0

//To see if the function correctly returns whichever value of n1 and n2 is //less than or equal to the other when one of the array size is 0

int v = findDifference(people, -1, bench, 6); //returns -1

//To see if the function correctly returns -1 is when the array size for

//any of the two array is negative

int w = findDifference(people, 5, bench, -1); //returns -1

//To see if the function correctly returns -1 is when the array size for

//any of the two array is negative

removeDups

string d[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

int p = removeDups(d, 9); // returns 5

// d[0] through d[4] now contain "clarence" "neil" "amy" "ketanji" "amy"

//To check if the function correctly returns the number of retained items //and correctly modifies the array so that the array from 0 index to

//#retained -1 has all the duplicates removed

string d[9] = {

"clarence", "amy", "amy", "amy", "dave", "ketanji", "ketanji", "amy", "amy"

};

int p = removeDups(d, 9); //returns 5

//To check if the function correctly returns the number of retained items //and correctly modifies the array so that the array from 0 index to

//#retained -1 has all the duplicates removed

string d[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

int p = removeDups(d, 4);

//To check if the function correctly returns the number of retained items

//of the size restricted array and modifies the array correctly so that

//from 0 index to #retained -1 has all the duplicates removed of the

//size restricted array

int q = removeDups(d, 0);

//To check if the function correctly returns 0 when the array is empty

//Also to check that it did not modify the array

int r = removeDups(d, -1);

//To check if the function correctly returns -1 when the array size is //negative. Also to check that it did not modify the array

subsequence

string big[10] = { "elena", "john", "amy", "ketanji", "neil", "amy" };

string little1[10] = { "john", "ketanji", "neil" };

bool u1 = subsequence(big, 6, little1, 3); // returns true

//To check if the function correctly returns true when a2 is a subsequence

//of a1

string little2[10] = { "amy", "john" };

bool u2 = subsequence(big, 6, little2, 2); // returns false

//To check if the function correctly returns false when a2 is not a subsequence of a1

string little3[10] = { "john", "amy", "amy" };

bool u3 = subsequence(big, 6, little3, 3); // returns true

//To check if the function correctly returns true when a2 is a subsequence

//of a1

string little4[10] = { "john", "john", "amy" };

bool u4 = subsequence(big, 6, little4, 3); // returns false

//To check if the function correctly returns false when a2 is not a subsequence of a1

bool u5 = subsequence(big, 6, little4, 0); // returns true

//To check if the function correctly returns true when a2 is an empty //array as every sequence, even a sequence of 0 elements, contains a //sequence of 0 elements.

bool u6 = subsequence(big, 0, little4, 0); //returns true

//To check if the function correctly returns true when both a1 and a2 is //an empty array as every sequence, even a sequence of 0 elements, //contains a sequence of 0 elements.

makeMerger

string x[5] = { "amy", "elena", "elena", "ketanji", "samuel" };

string y[4] = { "clarence", "elena", "john", "sonia" };

string z[20];

int n = makeMerger(x, 5, y, 4, z, 20); // returns 9

// z has amy clarence elena elena elena john ketanji samuel Sonia

//To see if the function correctly place in result all the elements //of a1 and a2 arranged in nondecreasing order and returns the number of //elements so placed

int o = makeMerger(x, 3, y, 2, z, 20); //returns 5

//z has amy clarence elena elena elena

//To see if the function correctly place in result first three elements of

//a1 and first two elements of a2 arranged in nondecreasing order and //returns the number of elements so placed

string x[5] = { "amy", "ketanji", "elena", "ketanji", "samuel" };

string y[4] = { "clarence", "elena", "john", "sonia" };

string z[20];

int n = makeMerger(x, 3, y, 2, z, 20); //returns -1

//To see if the function correctly returns -1 when elements of a1 is not //in nondecreasing order

string x[5] = { "amy", "elena", "elena", "ketanji", "samuel" };

string y[4] = { "elena", "clarence", "john", "sonia" };

string z[20];

int n = makeMerger(x, 3, y, 2, z, 20);

//To see if the function correctly returns -1 when elements of a2 is not //in nondecreasing order

string x[5] = { "amy", "ketanji", "elena", "ketanji", "samuel" };

string y[4] = { "elena", "clarence", "john", "sonia" };

string z[20];

int n = makeMerger(x, 3, y, 2, z, 20);

//To see if the function correctly returns -1 when elements of a1 and a2 //is not in nondecreasing order

string x[10] = { "amy", “amy”, “amy”, “amy”, “amy”, “amy”, "elena", "elena", "ketanji", "samuel" };

string y[11] = { "amy", “amy”, “amy”, “amy”, “amy”, “amy”, “bob”, "clarence", "elena", "john", "sonia" };

string z[20];

int n = makeMerger(x, 10, y, 11, z, 20);

//To see if the function correctly returns -1 when result would have more //than max elements

int n = makeMerger(x, 5, y, 5, z, 20);

//To see if the function correctly place in result first five elements of

//a1 and firstfive elements of a2 arranged in nondecreasing order and //returns the number of elements so placed (when both arrays combined //exceeds max)

int n = makeMerger(x, -1, y, 5, z, 20);

//To see if the function correctly returns -1 when one or both of the

//array size is negative

int n = makeMerger(x, 5, y, -1, z, 20);

//To see if the function correctly returns -1 when one or both of the

//array size is negative

int n = makeMerger(x, -1, y, -1, z, 20);

//To see if the function correctly returns -1 when one or both of the

//array size is negative

divide

string sc[6] = { "john", "amy", "samuel", "elena", "sonia", "neil" };

int x = divide(sc, 6, "ketanji"); // returns 3

//To see if the function correctly returns 3 and arranges the array so

//that elements that are < divider comes before all the other elements and //all the elements whose value is > divider comes after all the other

//elements

string sc2[4] = { "john", "sonia", "amy", "neil" };

int y = divide(sc2, 4, "neil");

//To see if the function correctly returns 2 and arranges the array so

//that elements that are < divider comes before all the other elements and //all the elements whose value is > divider comes after all the other

//elements

int z = divide(sc2, 4, "zoom");

//To see if the function correctly returns n if there are no such elements

//that is not < divider

int a = divide(sc2, 0, "neil");

//To see if the function correctly returns n if there are no such elements

//that is not < divider (since array size is 0 there are no such values //that is not < divider)

int b = divide(sc2, -1, "neil");

//To see if the function correctly returns -1 if passed a bad argument

//where array size is negative