**Challenges**

I had a bit of trouble coming up with an efficient solution to the separate function, and it took several tries to ensure that my function worked properly too. I also spent some time figuring out the boundary cases to each function that I implemented.

**Test Cases**

appendToAll:

string people[5] = { "alex", "brett", "charlie", "daniel", "edward" };

Append to entire array ( appendToAll(people, 5, "ly") )

Append to portion of array ( appendToAll(people, 2, "ly") )

Append to none of array (appendToAll(people, 0, "ly") )

Negative n value ( appendToAll(people, -1, "ly") )

Empty append string ( appendToAll(people, 5, "") )

lookup:

string fruit[6] = { "apple", "banana", "grapes", "watermelon", "mango", "orange" };

Lookup entire array with element contained ( lookup(fruit, 6, "grapes") )

Lookup entire array with element not contained ( lookup(fruit, 6, "kiwi") )

Lookup part of array with element contained ( lookup(fruit, 3, "grapes") )

Lookup part of array with element not contained ( lookup(fruit, 3, "mango") )

Lookup size zero array ( lookup(fruit, 0, "grapes") )

Different case ( lookup(fruit, 6, "Watermelon") )

Negative n value ( lookup(fruit, -1, "grapes") )

positionOfMax:

string colors[7] = { "blue", "orange", "red", "red", "magenta", "blue", "purple" };

Unique element as max ( positionOfMax(colors, 3) )

Duplicate elements as max ( positionOfMax(colors, 7) )

Size one array ( positionOfMax(colors, 1) )

Size zero array ( positionOfMax(colors, 0) )

Negative n value ( positionOfMax(colors, -1) )

rotateLeft:

string animals[6] = { "cat", "dog", "elephant", "giraffe", "panda", "penguin" };

Rotate first element ( rotateLeft(animals, 6, 0) )

Rotate middle element ( rotateLeft(animals, 6, 2) )

Rotate last element ( rotateLeft(animals, 6, 5) )

Rotate in one element array ( rotateLeft(animals, 1, 0) )

Rotate in zero element array ( rotateLeft(animals, 0, 0) )

Negative position ( rotateLeft(animals, 6, -1) )

Position greater than n ( rotateLeft(animals, 4, 5) )

countRuns:

string stationery[10] = { "pencil", "pencil", "pencil", "paper", "eraser", "pencil", "pencil", "folder", "binder", "eraser" };

Multiple runs ( countRuns(stationery, 10) )

No unique elements ( countRuns(stationery, 3) )

Size one array ( countRuns(stationery, 1) )

Size zero array ( countRuns(stationery, 0) )

Negative n value ( countRuns(stationery, -1) )

flip:

string furniture[5] = { "table", "chair", "desk", "lamp", "bed" };

Odd-sized segment to flip ( flip(furniture, 5) )

Even-sized segment to flip ( flip(furniture, 4) )

Flip one element ( flip(furniture, 1) )

Flip zero elements ( flip(furniture, 0) )

Negative n value ( flip(furniture, -1) )

differ:

string cities1[6] = { "San Francisco", "Los Angeles", "Chicago", "New York City", "Boston", "Seattle" };

string cities2[5] = { "San Francisco", "Los Angeles", "Sacramento", "San Jose", "San Diego" };

Check arrays with differences ( differ(cities1, 6, cities2, 5) )

Check arrays without differences ( differ(cities1, 3, cities2, 2) )

Size zero array ( differ(cities1, 0, cities2, 5) )

Negative n value ( differ(cities1, -1, cities2, 5) )

subsequence:

string majors1[5] = { "computer science", "math", "english", "biology", "business" };

string majors2[3] = { "math", "english", "history" };

string majors3[6] = { "computer science", "math", "english", "biology", "math", "english" };

Subsequence does not exist ( subsequence(majors1, 5, majors2, 3) )

Subsequence exists once ( subsequence(majors1, 5, majors2, 2) )

Subsequence exists more than once ( subsequence(majors3, 6, majors2, 2) )

Empty sequence to check for ( subsequence(majors1, 5, majors2, 0) )

Empty array to check ( subsequence(majors1, 0, majors2, 5) )

lookupAny:

string vegetables1[5] = { "lettuce", "asparagus", "spinach", "spinach", "cauliflower" };

string vegetables2[6] = { "broccoli", "green bean", "spinach", "cauliflower", "cucumber", "carrot" };

No common elements ( lookupAny(vegetables1, 2, vegetables2, 6) )

Multiple common elements ( lookupAny(vegetables1, 3, vegetables2, 6) )

Multiple of the same common element ( lookupAny(vegetables1, 5, vegetables2, 6) )

Negative n value ( lookupAny(vegetables1, -1, vegetables2, 6) )

Size zero array ( lookupAny(vegetables1, 5, vegetables2, 0) )

Size zero array ( lookupAny(vegetables1, 0, vegetables2, 6) )

separate:

string sports[5] = { "basketball", "football", "soccer", "baseball", "tennis" };

Separator not in array ( separate(sports, 5, "rugby") )

Separator in array ( separate(sports, 5, "basketball") )

Separate portion of array ( separate(sports, 2, "football") )

Separator would be greatest in array ( separate(sports, 5, "water polo") )

Separator would be smallest in array ( separate(sports, 5, "american football") )

Empty string separator ( separate(sports, 5, "") )

Negative n value ( separate(sports, -1, "soccer") )