1. Obstacles
   1. I first couldn’t figure out how to start the function countRuns so then instead of trying to count every new unique item in the array i instead just counted the items that were repeated consecutively and returned (n-the count).
   2. I was confused on the function subsequence, so I tried breaking it down step by step and first just found the index of the first item of a2 in a1 and then wrote the code using a new array. I then understood what was needed of the function better so I was abe to modify it so that creating a new array was no longer needed.
   3. I didnt know how to get started on the split function so I went for office hours, where I was better able to understand the function. After this I decided to just completely sort the array which fulfilled the requirements listed in the spec.
2. Test Cases

string people[5] = { "boris", "gordon", "rishi", "liz", "john" };

* 1. assert(appendToAll(people, 5, "!!!") == 5 && people[0] == "boris!!!" && people[4] == "john!!!"); //adds !!! to all elements
  2. assert(appendToAll(people, -5, "!!!") == -1); //for negative num
  3. assert(appendToAll(people, 0, "!!!") == 0 && people[0] == "boris!!!" && people[4] == "john!!!"); //when num of elements is 0
  4. assert(appendToAll(people, 1, "?") == 1 && people[0] == "boris!!!?" && people[4] == "john!!!"); //appends only n number of items

string people1[5] = { "boris", "gordon", "rishi", "liz", "john" };

* 1. assert(lookup(people1, 5, "gordon") == 1); //valid
  2. assert(lookup(people1, 1, "rishi") == -1); //checks that function only checks till n
  3. assert(lookup(people1, 5, "ris") == -1); //not in list
  4. assert(lookup(people1, -3, "go") == -1); //negative num
  5. assert(lookup(people1, 6, "john") == 4); //if repeated, returns smaller num

string pp2[6] = { "david", "liz", "margaret", "tony", "gordon", "boris" };

* 1. assert(positionOfMax(pp2, 6) == 3); //valid

string pp1[6] = { "tony", "liz", "margaret", "tony", "gordon", "boris" };

* 1. assert(positionOfMax(pp1, 6) == 0);//to check that it returns smallest if same
  2. assert(positionOfMax(pp1, 0) == -1); //returns -1 if number of items is 0
  3. assert(positionOfMax(pp1, -10) == -1);//when negative
  4. assert(positionOfMax(pp1, 1) == 0); //returns position if one element is there

string mp[5] = { "john", "david", "liz", "theresa", "margaret" };

* 1. assert((rotateLeft(mp, 5, 1) == 1) && mp[2] == "theresa" && mp[4] == "david");//valid

string mp1[5] = { "john", "david", "liz", "theresa", "margaret" };

* 1. assert((rotateLeft(mp1, 2, 1) == 1) && mp1[0] == "john"); //only considers n items

string mp2[5] = { "john", "david", "liz", "theresa", "margaret" };

* 1. assert((rotateLeft(mp2, 5, 4) == 4) && mp2[2] == "liz");//rotate last item to array is unchanged
  2. assert((rotateLeft(mp2, 5, 0) == 0) && mp2[4] == "john" && mp2[0] == "david");//first index
  3. assert(rotateLeft(mp2, -2, 0) == -1); //when negative num
  4. assert(rotateLeft(mp2, 5, -4) == -1); //when pos negative
  5. assert(rotateLeft(mp2, 5, 5) == -1); //when pos doesnt exist in array

string d[5] = { "margaret", "margaret", "margaret", "tony", "tony" };

* 1. assert(countRuns(d, 5) == 2); //valid

string d1[9] = {"tony", "tony", "rishi", "rishi", "gordon", "gordon", "gordon", "rishi", "tony"};

* 1. assert(countRuns(d1, 9) == 5); //checks for repeated words non sequentially
  2. assert(countRuns(d1, 2) == 1); //only one run
  3. assert(countRuns(d1, 3) == 2); //2 runs
  4. assert(countRuns(d1, -8) == -1); //negative num
  5. assert(countRuns(d1, 0) == 0); //for 0
  6. assert(countRuns(d1, 1) == 1); //for n as 1

string d2[9] = { "tony", "boris", "rishi", " ", " ", "gordon", "gordon", "rishi", "-"};

* 1. assert(countRuns(d2, 9) == 7); // to check for space and non alplabets

string leader[6] = { "boris", "rishi", "", "tony", "theresa", "david" };

* 1. assert((flip(leader, 4) == 4) && leader[0] == "tony" && leader[5] == "david"); //only for n items
  2. assert((flip(leader, 0) == 0) && leader[0] == "tony"); //for 0
  3. assert(flip(leader, -4) == -1); //negative num

string leader1[6] = { "boris", "rishi", "", "tony", "theresa", "david" };

string politician[5] = { "boris", "rishi", "david", "", "tony" };

* 1. assert(differ(leader1, 6, politician, 5) == 2); //valid
  2. assert(differ(leader1, 2, politician, 1) == 1); // check for n1 and n2
  3. assert(differ(leader1, -2, politician, 1) == -1); //negative n1
  4. assert(differ(leader1, 0, politician, 1) == 0); //n1 < n2
  5. assert(differ(leader1, 2, politician, 5) == 2); //n1 > n2

string h[7] = { "rishi", "margaret", "gordon", "tony", "", "john", "liz" };

string e[4] = { "gordon", "tony", "", "john" };

* 1. assert(subsequence(h, 7, e, 4) == 2);//valid
  2. assert(subsequence(h, -3, e, 4) == -1);//negative n1
  3. assert(subsequence(h, 7, e, -5) == -1);//negative n2

string names[10] = { "john", "margaret", "theresa", "rishi", "boris", "liz", "john", "margaret" };

string names1[10] = { "margaret", "theresa", "rishi" };

* 1. assert(subsequence(names, 6, names1, 3) == 1); //valid

string names2[10] = { "boris", "liz", "rishi" };

* 1. assert(subsequence(names, 6, names2, 3) == -1); //not all elements are repeated

string names3[10] = { "john", "rishi" };

* 1. assert(subsequence(names, 5, names3, 2) == -1); //names arent in the right order in a1

string names4[10] = { "john", "margaret" };

* 1. assert(subsequence(names, 8, names4, 2) == 0); //subsequence appears more than once
  2. assert(subsequence(names, 0, names4, 0) == 0); //edge

string namess[10] = { "john", "margaret", "theresa", "rishi", "boris", "liz" };

string set1[10] = { "david", "boris", "rishi", "margaret" };

* 1. assert(lookupAny(namess, 6, set1, 4) == 1); // valid

string set2[10] = { "tony", "gordon" };

* 1. assert(lookupAny(namess, 6, set2, 2) == -1); // no matches

string set3[10] = { "boris", "liz" };

* 1. assert(lookupAny(namess, 6, set3, 2) == 4); //2 names match

string namess1[10] = { "john", "john", "theresa", "john", "boris", "liz" };

string set4[10] = { "john", "john" };

* 1. assert(lookupAny(namess1, 6, set4, 2) == 0); //multiple same matches
  2. assert(lookupAny(namess1, -5, set4, 0) == -1); //negative n1
  3. assert(lookupAny(namess1, 2, set4, -5) == -1); //negative n2
  4. assert(lookupAny(namess1, 5, set4, 0) == -1);//n2 0

string pm[6] = { "david", "liz", "margaret", "tony", "gordon", "boris" };

* 1. assert(split(pm, 6, "john") == 3); //when string isnt in the array

string pm2[4] = { "margaret", "theresa", "liz", "rishi" };

* 1. assert(split(pm2, 4, "rishi") == 2); //string in the last item in the array

string pm3[10] = { "f", "b", "b", "a", "e", "aa", "p", "k", "g", "d" };

* 1. assert(split(pm3, 10, "z") == 10); //no larger item exists in the array

string pm4[10] = { "f", "b", "b", "a", "e", "aa", "p", "k", "g", "d" };

* 1. assert(split(pm4, 6, "a") == 0); //splitter is the smallest item in the array

string pm5[10] = { "f", "b", "b", "a", "e", "aa", "p", "k", "g", "d" };

* 1. assert(split(pm5, 2, "p") == 2); //splitter is in the array
  2. assert(split(pm5, -4, "p") == -1); //negative num
  3. assert(split(pm5, 0, "p") == 0); //n is 0