1. One obstacle I faced while doing this project was trying to figure out when to break the loop, especially for functions that modified the order of elements in the array, such as separate and subsequence. I solved this by keeping count of how many times the looped as a separate, and terminating the loop once it reached n. I also couldn’t think of a way to do countRuns and separate, but I just broke the problem down simpler to make it easier to solve.

|  |  |
| --- | --- |
| string h[7] = { "mick", "marie" , "fiona", "rudy", "", "gordon", "lindsey" };  appendToAll(h, 7, “!!!”); | **Testing function** |
| appendToAll(h, -2, “!!!”); | **N is negative** |
| appendToAll(h, 5, “!!!”); | **N is less than the number of elements** |
| string h[7] = { "mick", "marie" , "fiona", "rudy", "", "gordon", "lindsey" };  lookup(h, 7, “rudy”); | Testing function |
| Lookup(h, 7, “adam”); | Target is not in array |
| Lookup(h, 5, “lindsey”); | N is less than number of elements |
| Lookup(h, 7, “Lindsey”); | Target not in array, but same word with different case |
| Lookup(h, -2, “rudy”); | N is negative |
| string h[7] = { "mick", "marie" , "fiona", "rudy", "rudy", "gordon", "lindsey" };  positionOfMax(h, 7); | **Testing function** |
| positionOfMax(h, -2); | **N is negative** |
| positionOfMax(h, 2); | **N is less than number of elements** |
| string h[7] = { "mick", "marie" , "fiona", "rudy", "", "gordon", "lindsey" };  positionOfMax(h, 7); | **Same max string appears multiple times** |
| rotateLeft(h, -2, 5); | N is negative |
| string h[7] = { "mick", "marie" , "fiona", "rudy", "", "gordon", "lindsey" };  rotateLeft(h, 7, 3); | Testing function |
| countRuns(h, -2); | **N is negative** |
| string h[9] = { “rudy”, “adam”, “mike”, “mike”, “fiona”, “fiona”, “fiona”, “mike”, “mike” };  countRuns(h, 9); | **Testing function** |
| string h[4] = { "rudy", "adam", "mike", "fiona" };  countRuns(h, 4); | **All unique strings** |
| string h[4] = { "rudy", "rudy", "rudy", "rudy" };  countRuns(h, 4); | **All identical strings** |
| string h[9] = { "rudy", "adam", "mike", "mike", "fiona", "fiona", "fiona", "mike", "mike" };  countRuns(h, 6); | **N less than number of elements** |
| string h[9] = { "rudy", "adam", "mike", "mike", "Fiona", "fiona", "fiona", "mike", "mike" };  countRuns(h, 9); | **Word appears in middle of run, but with different case** |
| string h[7] = { "mick", "marie" , "fiona", "rudy", "", "gordon", "lindsey" };  flip(h, 7); | Testing function |
| Flip(h, -2); | N is negative |
| Flip(h, 1); | Only one element in array |
| string folks[7] = { “adam”, “”, “fiona”, “mike”, “rudy”, “nancy”, “donald" };  string group[6] = {“adam”, “”, “fiona”, “donald", “mike”, “rudy”};  int r = differ(folks, 7, group, 6); | **Testing function** |
| Differ(folks, -2, group, 6); | **n is negative** |
| Differ(folks, 1, group, 2); | **N1 less than number of elements** |
| Differ(folks, 7, group, 7); | **Identical arrays** |
| string folks[7] = { “adam”, “”, “fiona”, “mike”, “rudy”, “nancy”, “donald" };  string group[6] = {“adam”, “”, “fiOna”, “donald", “mike”, “rudy”};  int r = differ(folks, 7, group, 7); | **same word, different case** |
| string names[10] = {“gordon", “marie”, “nancy”, “mick”, “adam”, “lindsey”};  string names1[10] = { “marie”, “nancy”, “mick”};  int u = subsequence(names, 6, names1, 3); | Testing function |
| Subsequence(names, -2, names1, 3); | N is negative |
| Subsequence(names, 3, names1, 3); | N less than number of elements |
| string names[10] = { "gordon", "marie", "nancy", "mick", "marie", "nancy", "mick" };  string names1[10] = { "marie", "nancy", "mick"};  int u = subsequence(names, 7, names1, 3); | Sequence appears multiple times |
| string names[10] = { "gordon", "adam", "lindsey"};  string names1[10] = { "marie", "nancy", "mick"};  int u = subsequence(names, 3, names1, 3); | Does not contain sequence |
| string names[10] = {“gordon", “marie”, “nancy”, “mick”, “adam”, “lindsey”};  string set1[10] = {“donald", “adam”, “mick”, “marie”};  lookupAny(names, 6, set1, 4); | **Testing function** |
| lookupAny(names, -2, set1, 4); | **n is negative** |
| string names[10] = { "gordon", "marie", "nancy", "mick", "adam", "lindsey"};  string set1[10] = {"donald", "alex", "mike", "bob"};  lookupAny(names, 6, set1, 4); | **All array elements are different** |
| lookupAny(names, 5, set1, 4); | **N is less than number of elements** |
| string names[10] = {“gordon", “marie”, “nancy”, “mick”, “adam”, “lindsey”};  string set1[10] = {“donald", “adam”, “mick”, “marie”};  lookupAny(names, 6, set1, 4); | **Case difference in a1** |
| string names[10] = {“gordon", “marie”, “nancy”, “mick”, “adam”, “lindsey”};  string set1[10] = {“donald", “adam”, “mick”, “Marie”};  LookupAny(names, 6, set1, 4); | **case difference in a2** |
| string persons[6] = {“donald", “lindsey”, “marie”, “rudy”, “fiona”, “adam”};  separate(persons, 6, “gordon"); | Testing function |
| Separate(persons, -2, “gordon”); | N is negative |
| Separate(persons, 4, “gordon”); | N less than number of elements |