**Project 3: CS31**

**Obstacles I overcame:**

One notable obstacle I overcame while writing my program for this assignment was the fact that a lot of the functions we needed to write required a logical breakdown of how to solve the problem rather than being a computational problem. For functions such as the subsequence or the rotateleft function especially, I spent more time on the conceptual way of breaking down the problem rather than the computational method of implementing the function, as most of these utilized fairly similar functions (*for* *loops, if-else statements*). Another notable obstacle I overcame while writing my program for this assignment was making sure that there were no subscripts that were attempting to access past the array size. Especially in this type of project, where there are many of the same types of methods for creating the function, it is really easy to mess up and miss a mistake that would access past the array size. I overcame this by going through my code methodically using the “desk-checking” idea in order to make sure there were no mistakes in my code.

**Test Data:**

*appendToAll Function*

|  |  |
| --- | --- |
| Array: people[5] = { "donald", "lindsey", "fiona", "rudy", "mick" };  appendToAll(people, 5, "!!!"); | Makes sure function works correctly by appending “!!!” onto the array |
| Array: people[5] = { "donald", "lindsey", "fiona", "rudy", "mick" };  appendToAll(people, -5, "!!!"); | If the n value is negative |
| Array: people[5] = { "donald", "lindsey", "fiona", "rudy", "mick" };  appendToAll(people, 3, "!!!"); | If the array is bigger than the provided n value |
| Array: people[5] = { "donald", "lindsey", "fiona", "rudy", "mick" };  appendToAll(people, 0, "!!!"); | If n is 0 |

*lookup Function*

|  |  |
| --- | --- |
| Array: officeholders[5] = { "donald", "lindsey", "mike", "adam", "nancy" };  lookup(officeholders, 5, "adam"); | Makes sure function works correctly by finding the position of a string in the array that is equal to target |
| Array: officeholders[5] = { "donald", "lindsey", "mike", "adam", "nancy" };  lookup(officeholders, -5, "adam"); | If the n value is negative |
| Array: officeholders[5] = { "donald", "lindsey", "mike", "adam", "nancy" };  lookup(officeholders, 5, "gordon"); | If the string is not present in the array |
| Array: officeholders[5] = { "adam", "lindsey", "mike", "adam", "nancy" };  lookup(officeholders, 5, "adam"); | If there are multiple of the same string in the array |
| Array: officeholders[5] = { "donald ", "lindsey", "mike", "adam", "nancy" };  lookup(officeholders, 3, "adam"); | If the array is bigger than the provided n value |
| Array: officeholders[5] = { "donald ", "lindsey", "mike", "adam", "nancy" };  lookup(officeholders, 0, "adam"); | If n is 0 |

*positionOfMax Function*

|  |  |
| --- | --- |
| Array: persons[6] = { "donald", "lindsey", "marie", "rudy", "fiona", "adam" };  positionOfMax(persons, 6); | Makes sure function works correctly by finding the position of a string that is the biggest |
| Array: persons[6] = { "donald", "lindsey", "marie", "rudy", "fiona", "adam" };  positionOfMax(persons, -6); | If the n value is negative |
| Array: persons[6] = { "donald", "rudy", "rudy", "rudy", "fiona", "adam" };  positionOfMax(persons, 6); | If the same string is present multiple times in the array |
| Array: persons[6] = { "adam", "adam", "adam", "adam", "adam", "adam" };  positionOfMax(persons, 6); | If all the strings are of the same size |
| Array: persons[6] = { "donald", "lindsey", "marie", "rudy", "fiona", "adam" };  positionOfMax(persons, 3); | If the array is bigger than the provided n value |
| Array: persons[6] = { "donald", "lindsey", "marie", "rudy", "fiona", "adam" };  positionOfMax(persons, 0); | If n is 0 |

*rotateLeft Function*

|  |  |
| --- | --- |
| Array: politician[5] = { "mike", "donald", "lindsey", "nancy", "adam" };  rotateLeft(politician, 5, 1); | Makes sure function works correctly rotating all the strings to the left and moving the string at the position to the end |
| Array: politician[5] = { "mike", "donald", "lindsey", "nancy", "adam" };  rotateLeft(politician, -5, 1); | If the n value is negative |
| Array: politician[5] = { "mike", "donald", "lindsey", "nancy", "adam" };  rotateLeft(politician, 5, -1); | If position value is less than 0 or bigger than or equal to the n value |
| Array: politician[5] = { "mike", "donald", "lindsey", "nancy", "adam" };  rotateLeft(politician, 0, 1); | If n is 0 |
| Array: politician[5] = { "mike", "donald", "lindsey", "nancy", "adam" };  rotateLeft(politician, 3, 0); | If the array is bigger than the provided n value |

*countRuns Function*

|  |  |
| --- | --- |
| Array: d[9] = {"rudy", "adam", "mike", "mike", "fiona", "fiona", "fiona", "mike", "mike"};  countRuns(d, 9); | Makes sure function works correctly by returning number of sequences |
| Array: d[9] = {"rudy", "adam", "mike", "mike", "fiona", "fiona", "fiona", "mike", "mike"};  countRuns(d, -9); | If the n value is negative |
| Array: d[9] = {"rudy", "adam", "mike", "mike", "fiona", "fiona", "fiona", "mike", "mike"};  countRuns(d, 0); | If n is 0 |
| Array: d[6] = { "adam", "adam", "adam", "adam", "adam", "adam" };  countRuns(d,6); | If all the values in the array are the same |
| Array: d[9] = {"rudy", "adam", "mike", "mike", "fiona", "fiona", "fiona", "mike", "mike"};  countRuns(d, 6); | If the array is bigger than the provided n value |

*flip Function*

|  |  |
| --- | --- |
| Array: folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" };  int q = flip(folks, 5); | Makes sure function works correctly by reversing the order of the elements & If the array is bigger than the provided n value |
| Array: folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" };  int q = flip(folks, -5); | If the n value is negative |
| Array: folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" };  int q = flip(folks, 0); | If n is 0 |

*differ Function*

|  |  |
| --- | --- |
| Array1: folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" };  Array2: group[6] = { "adam", "", "fiona", "donald", "mike", "rudy" };  differ(folks, 7, group, 6); | Makes sure function works correctly by returning the first spot of differences between the two arrays & If the array is bigger than the provided n value |
| Array1: folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" };  Array2: group[6] = { "adam", "", "fiona", "donald", "mike", "rudy" };  differ(folks, -7, group, -6); | If either n value is negative or both are negative |
| Array1: folks[7] = { "adam", "", "fiona", "mike", "rudy", "nancy", "donald" };  Array2: group[6] = { "adam", "", "fiona", "donald", "mike", "rudy" };  differ(folks, 0, group, 0); | If either n is 0 or both are 0 |
| Array1: folks[7] = { "adam", "", "fiona", "mike"};  Array2: group[6] = { "adam", "", "fiona"};  differ(folks, 3, group, 3); | If n1 and n2 are the same size but there are no differences |
| Array1: folks[7] = { "adam", "", "fiona", "mike"};  Array2: group[6] = { "adam", "", "fiona"};  differ(folks, 2, group, 3); | If n1 is smaller than n2 but there are no differences |
| Array1: folks[7] = { "adam", "", "fiona", "mike"};  Array2: group[6] = { "adam", "", "fiona"};  differ(folks, 3, group, 2); | If n2 is smaller than n1 but there are no differences |

*subsequence Function*

|  |  |
| --- | --- |
| Array1: names[10] = {"gordon", "marie", "nancy", "mick", "adam", "lindsey"};  Array2: names1[6] = {"marie", "nancy", "mick"};  subsequence(names, 6, names1, 3); | Makes sure function works correctly by identifying if the subsequence exists & If the array is bigger than the provided n value |
| Array1: names[10] = {"gordon", "marie", "nancy", "mick", "adam", "lindsey"};  Array2: names1[6] = {"marie", "nancy", "mick"};  subsequence(names, -6, names1, -3); | If either n value is negative or both are negative |
| Array1: names[10] = {"gordon", "marie", "nancy", "mick", "adam", "lindsey"};  Array2: names1[6] = {"marie", "nancy", "mick"};  subsequence(names, 0, names1, 0); | If either n is 0 or both are 0 |
| Array1: names[10] = {"gordon", "marie", "nancy", "mick", "adam", "lindsey"};  Array2: names1[6] = {"donald”};  subsequence(names, 6, names1, 1); | If the sequence is not present in the first array |
| names[10] = { "gordon", "marie", "nancy", "mick", "adam", "marie", "nancy", "mick" };  names1[10] = { "marie", "nancy", "mick" };  int t = subsequence(names, 8, names1, 3); | If the second array sequence is present multiple times in the first array |
| names[10] = { "gordon", "marie", "nancy", "donald", "mick"};  names1[10] = { "marie", "nancy", "mick" };  int t = subsequence(names, 8, names1, 3); | If the sequence is present in the first array from the second array but is not contiguous |

*lookupAny Function*

|  |  |
| --- | --- |
| Array1: names[10] = { "gordon", "marie", "nancy", "mick", "adam", "lindsey" };  Array2: set1[10] = { "donald", "adam", "mick", "marie" };  lookupAny(names, 2, set1, 4); | Makes sure function works correctly by finding which values are the same in the two arrays & If the array is bigger than the provided n value |
| Array1: names[10] = { "gordon", "marie", "nancy", "mick", "adam", "lindsey" };  Array2: set1[10] = { "donald", "adam", "mick", "marie" };  lookupAny(names, -2, set1, -4); | If either n value is negative or if both are negative |
| Array1: names[10] = { "gordon", "marie", "nancy", "mick", "adam", "lindsey" };  Array2: set1[10] = { "donald", "adam", "mick", "marie" };  lookupAny(names, 0, set1, 0); | If either n is 0 or both are |
| Array1: names[10] = { "donald", "donald", "nancy", "mick", "adam", "lindsey" };  Array2: set1[10] = { "bob", "adam", "mick", "donald" };  lookupAny(names, 0, set1, 0); | If there are multiple of the same string in the first array, it should return the first position of equality in array 1 between the two arrays |

*separate Function*

|  |  |
| --- | --- |
| Array: persons[6] = {"donald", "lindsey", "marie", "rudy", "fiona", "adam" };  separate(persons, 6, "gordon"); | Makes sure that the function works the way it is supposed to by separating the names properly |
| Array: persons[6] = {"donald", "gordon", "marie", "rudy", "fiona", "adam" };  separate(persons, 6, "gordon"); | If separator string is in the array |
| Array: persons[6] = {"donald", "lindsey", "gordon", "gordon", "fiona", "adam" };  separate(persons, 6, "gordon"); | If separator string is present multiple times in the array |
| Array: persons[6] = {"rudy", "rudy", "rudy", "rudy", "rudy", "rudy" };  separate(persons, 6, "gordon"); | If all the terms of the array are larger than the separator |
| Array: persons[6] = {"fiona", "fiona", "fiona", "fiona", "fiona", "fiona"};  separate(persons, 6, "gordon"); | If all the terms of the array are smaller than the separator |
| Array: persons[6] = {"donald", "lindsey", "marie", "rudy", "fiona", "adam" };  separate(persons, -6, "gordon"); | If the n value is negative |
| Array: persons[6] = {"donald", "lindsey", "marie", "rudy", "fiona", "adam" };  separate(persons, 0, "gordon"); | If the n value is 0 |