**Project 4 Report**

Notable Obstacles:

The main obstacle I encountered in project 4 was coming up with the logic to create my program. Just like what I did in project 3, I began writing pseudocode for my functions. The pseudocode made it much easier to lay out my thought process for a function. The hardest function for me to write was the split function. I wrote three different pseudocodes for the split function. I was trying to write it without using a second array. After several hours of trying and failing, I was finally able to do so. I was able to do so by sorting the array alphabetically.

Test Data:

string cities[7] = { “millbrae”, “burlingame”, “sunnyvale”, “cupertino”, “belmont”, “berkeley”, “oakland” };

* appendToAll(cities, -4, “!!!”)
  + Should return -1
  + Reason: to test if the function can successfully determine that n is negative
* appendToAll(cities, 7, “??”)
  + Should return 7
  + Reason: to test if the function can successfully return n, which in this case is 7, and appends “??” to the end of each element in cities
* lookup(cities, -2, “cupertino”)
  + Should return -1
  + Reason: to test if the function can successfully determine that n is negative
* lookup(cities, 7, “berkeley”)
  + Should return 5
  + Reason: to test if the function can successfully determine that an element in the array is equal to the target and return that element’s position in the array
* lookup(cities, 7, “millbrae”)
  + Should return 0
  + Reason: to test if the function can successfully determine that an element in the array is equal to the target and return that element’s position in the array
* lookup(cities, 7, “hillsborough”)
  + Should return -1
  + Reason: to test if the function can successfully determine that there is no element in the array that is equal to the target and return -1

string empty[4] = { };

* positionOfMax(cities, -5)
  + Should return -1
  + Reason: to test if the function can successfully determine that n is negative
* positionOfMax(empty, 0)
  + Should return -1
  + Reason: to test if the function can successfully return -1 if the array has no elements
* positionOfMax(cities, 7)
  + Should return 2
  + Reason: to test if the function can successfully determine the largest string in the array and return its position
* rotateLeft(cities, -2, 3)
  + Should return -1
  + Reason: to test if the function can successfully determine that n is negative
* rotateLeft(cities, 7, -2)
  + Should return -1
  + Reason: to test if the function can successfully determine that pos is negative
* rotateLeft(cities, 7, 1)
  + Should return 1
  + Reason: to test if the function can successfully eliminate the element at the specified position (in this case 1), copy the elements after the specified position one place to the left, put the element that was eliminated into the last position of the array, and return the original position of the element that was eliminated from its original position and moved to the end of the array
* rotateLeft(cities, 7, 4)
  + Should return 4
  + Reason: to test if the function can successfully eliminate the element at the specified position (in this case 4), copy the elements after the specified position one place to the left, put the element that was eliminated into the last position of the array, and return the original position of the element that was eliminated from its original position and moved to the end of the array

string repeats[10] = { “millbrae”, “millbrae”, “millbrae”, “millbrae”, “burlingame”, “burlingame”, “cupertino”, “cupertino”, “cupertino”, “belmont” };

* countRuns(repeats, -6)
  + Should return -1
  + Reason: to test if the function can successfully determine that n is negative
* countRuns(repeats, 10)
  + Should return 4
  + Reason: to test if the function can successfully determine the number of sequences of one or more consecutive identical items in the array and return that number

string cities2[4] = {“millbrae”, “burlingame”, “cupertino”, “berkeley” };

* flip(cities2, -4)
  + Should return -1
  + Reason: to test if the function can successfully determine that n is negative
* flip(cities2, 4)
  + Should return 4
  + Reason: to test if the function can successfully reverse the order of the elements of the array and return n

string cities3[6] = {“millbrae”, “burlingame”, “sunnyvale”, “cupertino”, “belmont”, “oakland”};

string cities4[5] = {“millbrae”, “burlingame”, “sunnyvale”, “oakland”, “berkeley”};

* differ(cities3, -6, cities4, 5)
  + Should return -1
  + Reason: to test if the function can successfully determine that n1 is negative
* differ(cities3, 6, cities4, -5)
  + Should return -1
  + Reason: to test if the function can successfully determine that n2 is negative
* differ(cities3, 6, cities4, 5)
  + Should return 3
  + Reason: to test if the function can successfully determine and return the position of the first corresponding elements of a1 and a2 that are not equal if n1 > n2
* differ(cities3, 2, cities4, 1)
  + Should return 1
  + Reason: to test if the function can successfully determine and return the position of the first corresponding elements of a1 and a2 that are not equal if n1 > n2
* differ(cities4, 5, cities3, 6)
  + Should return 3
  + Reason: to test if the function can successfully determine and return the position of the first corresponding elements of a1 and a2 that are not equal if n2 > n1

string cities5[6] = {“millbrae”, “burlingame”, “sunnyvale”, “cupertino”, “belmont”, “berkeley”};

string cities6[3] = { “sunnyvale”, “cupertino”, “belmont” };

string cities7[2] = { “millbrae”, “berkeley” }

* subsequence(cities5, -6, cities6, 3)
  + Should return -1
  + Reason: to test if the function can successfully determine that n1 is negative
* subsequence(cities5, 6, cities6, -3)
  + Should return -1
  + Reason: to test if the function can successfully determine that n2 is negative
* subsequence(cities5, 6, cities6, 3)
  + Should return 2
  + Reason: to test if the function can successfully determine that all n2 elements in a2 appear in a1, consecutively and in the same order, and return the position in a1 where the subsequence beings
* subsequence(cities5, 4, cities7, 2)
  + Should return -1
  + Reason: to test if the function can successfully determine that the elements in a2 do not appear consecutively and in the same order in a1 and return -1

string cities8[5] = { “hillsborough”, “arcadia”, “chicago”, “millbrae”, “cupertino” };

string cities9[2] = { “boston”, “fremont” };

* lookupAny(cities, -7, cities8, 5)
  + Should return -1
  + Reason: to test if the function can successfully determine that n1 is negative
* lookupAny(cities, 7, cities8, -5)
  + Should return -1
  + Reason: to test if the function can successfully determine that n2 is negative
* lookupAny(cities, 7, cities8, 5)
  + Should return 0
  + Reason: to test if the function can successfully determine and return the position in a1 of an element that is equal to any of the elements in a2
* lookupAny(cities, 7, cities9, 2)
  + Should return -1
  + Reason: to test if the function can successfully determine that none of the elements in a2 appear in a1 and return -1
* split(cities, -7, “hillsborough”)
  + Should return -1
  + Reason: to test if the function can successfully determine that n is negative
* split(cities, 7, “hillsborough”)
  + Should return 4
  + Reason: to test if the function can successfully rearrange the elements of the array so that elements whose values is less than splitter come before all the other elements and all the elements whose value is larger than splitter come after all the other elements and return the position of the first element that, after the arrangement, is not less than splitter
* split(cities, 7, “woodside”)
  + Should return 7
  + Reason: to test if the function can successfully rearrange the elements of the array so that elements whose values is less than splitter come before all the other elements and all the elements whose value is larger than splitter come after all the other elements and return n, since there are no elements greater than “woodside” in the cities array