1. The only notable obstacle I had with this program was creating the split function. At first, I assumed that creating this function would require me to code an insertion sort algorithm using the rotate left function that I had just created. However, I realized that it wasn’t as simple as that because I could get stuck in an infinite loop or if I had a element equal to the splitter, it could end up being in the wrong place in the reorganized array. If was especially hard to write this function without creating a new array, but with some extra variables to reposition the array if the splitter was actually in the array, I was able to overcome this obstacle and create a working function.
2. string hero[6] = { "clark", "peter", "reed", "tony", "diana", "bruce" };

string hero2[5] = { "reed", "sue", "peter", “sue”, "steve" };

string d[9] = {  
 "tony", "bruce", "steve", "steve", "diana", "diana", "diana", "steve", "steve"  
};

string folks[6] = { "bruce", "steve", "", "tony", "sue", "clark" };  
string group[5] = { "bruce", "steve", "clark", "", "tony" };

string names[10] = { "logan", "reed", "sue", "selina", "bruce", "peter", “reed”, “selina”, “sue” };  
string names1[10] = { "reed", "sue", "selina" };

string names2[10] = { "logan", "selina" };

* Adding string to all elements of an array: appendToAll(hero, 6, “!!!”)
* Adding string to some elements of an array: appendToAll(hero, 3, “!!!”)
* Adding string to no elements of an array: appendToAll(hero, 0, “!!!”)
* Negative n value: appendToAll(hero, -4, “!!!”)
* Looking for a value in an array: lookup(hero, 6, “bruce”)
* Looking for a value that appears twice in an array: lookup(hero2, 5, “sue”)
* Looking for a value in part of an array: lookup(hero, 4, “reed”)
* Looking for a value that isn’t in an array: lookup(hero, 6, “brady”)
* Looking for a value in no parts of an array: lookup(hero, 0, “clark”)
* Negative n value: lookup(hero, -6, “clark”)
* Looking for the largest value in an array: positionOfMax(hero, 6);
* Looking for the largest value in an array if it appears twice: positionOfMax(hero2, 5)
* Looking for the largest value in part of an array: positionOfMax(hero, 3);
* Looking for the largest value in no parts of an array: positionOfMax(hero, 0);
* Negative n value: positionOfMax(hero, -5)
* Rotate everything to the left: rotateLeft(hero, 6, 0)
* Rotate everything to the left from the middle: rotateLeft(hero, 6, 3)
* Rotate everything to the left from the end: rotateLeft(hero, 6, 5)
* Position is equal to or greater than n: rotateLeft(hero, 6, 6)
* Negative n value: rotateLeft(hero, -4, 4)
* Negative pos value: rotateLeft(hero, 6, -4)
* Counts number of sequences of consecutive elements: countRuns(d, 9)
* Counts number of sequences of consecutive elements in part of the array: countRuns(d, 6)
* Count number of sequences of consecutive elements in no parts of the array: countRuns(d, 0)
* Negative n value: countRuns(d, -4)
* Flip entire array: flip(hero, 6)
* Flip odd length array: flip(hero2, 5)
* Flip part of an entire array: flip(hero, 4)
* Flip no part of an array: flip(hero, 0)
* Negative n value: flip(hero, -4)
* Compare two arrays for difference: differ(folks, 6, group, 5)
* Compare part of two arrays for difference: differ(folks, 4, group 3)
* When there is no difference: differ(folks, 2, group, 1)
* Either n value is 0: differ(folks, 0, group, 1)
* If both n values are 0: differ(folks, 0, group, 0;
* Either n value is negative: differ(folks, -2, group, 1)
* Subsequence exists in the array: subsequence(names, 6, names1, 3)
* Subsequence doesn’t exist in the array: subsequence(names, 5, names2, 2;
* Subsequence exists twice in the array: subsequence(names, 9, names1, 3)
* Subsequence logic error if n1 is less than n2: subsequence(names, 0, names1, 3)
* Either n value is negative: subsequence(names, -4, names1, -3)
* Either n value is zero: subsequence(names, 4, names1, 0)
* Contains any of the strings: lookupAny(hero, 6, hero2, 5)
* Doesn’t contain any of the strings: lookupAny(hero, 6,names2, 2)
* Either n is 0: lookupAny(hero, 0, heroes2, 2)
* Either n is negative: lookupAny(hero, -2, heroes2, 4)
* Split an array that doesn’t have splitter: split(hero, 6, "logan")
* Split an array that has splitter: split(hero, 6, “diana”)
* N is 0: split(hero, 0, “diana”)
* N is negative: split(hero, -2, “diana”)