* A brief description of notable obstacles you overcame.
* The most challenging obstacles that I encountered in Project 4 was the algorithm for the function *int divide(string a[], int n, string divider)*. First I took the sorting move, but I realized the I might need to create another array to store the correct order. But then the mechanics of conveyor belt inspired me to use rotateLeft() as a helper function to deals with items that were greater than the divider, because I could simply just move them to the back of the arrays. The idea was nice, but soon I realized it is not that simple. It took me a while to discover and observed the appropriate conditions for the while loop. Then the second obstacles I had was the situation where the divider was one of the elements in the array, but after many different code tracings, I eventually came up with two if statement that can check and avoid the issues when such situations came up.
* A list of the test data that could be used to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.
* Arrays set::
* string append[5] = {"WHAT", "WHO", "WHY"};
* string h[8] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona", "ed" };
* string max[5] ={"1","2","3","4","4"};
* string folks[6] = { "betty", "john", "", "xavier", "kevin", "dianne" };
* string politician[5] = { "eleni", "dianne", "fiona", "kevin", "gavin" };
* string k[9] = {"xavier", "betty", "john", "john", "ed", "ed", "ed", "john", "john"};
* string alpa[11] ={"A", "B","B","C","C","C","c","D","D","D","d"}; // A B C
* string folk[6] = { "betty", "john", "", "xavier", "kevin", "dianne" };
* string folkss[6] = { "betty", "john", "", "xavier", "kevin", "dianne" };
* string groups[5] = { "betty", "john", "dianne", "", "xavier" };
* string names[10] = { "eleni", "gavin", "kevin", "greg", "betty", "fiona" };
* string names1[10] = { "gavin", "kevin", "greg" };
* string r1[5]={"1","2","3","1","2"};
* string r2[2] ={"1","2"};
* string name[10] = { "eleni", "gavin", "kevin", "greg", "betty", "fiona" };
* string set1[10] = { "dianne", "betty", "greg", "gavin" };
* string randOrder[9]={"6","5","7","9","3","4","1","2","8"};
* string randOrder1[6]={"3","5","4","9","2"};
* Normal adding appendToAll(append, 3, "?!!!");
* Adding to 0 element appendToAll(append, 0, "?!!!");
* Adding to only first element appendToAll(append, 1, "?!!!");
* Look for exist element assert(lookup(h, 7, "eleni") == 5);
* Look for exit element that appears more than once assert(lookup(h, 7, "ed") == 2);
* Look for element that is not in the array assert(lookup(h, 2, "ed") == -1);
* Look for element of 0 element assert(lookup(h, 2, "ed") == -1);
* return -1 if there is no interesting element assert(positionOfMax(max, 0) == -1);
* //return smallest pos if there is two max assert(positionOfMax(max, 5) == 3);
* return -1 when there is 0 element to rotate
* assert(rotateLeft(folks, 0, 1) == -1);
* rotate left only first five element rotateLeft(folks, 5, 3) == 3
* rotate everything left rotate left only first five element rotateLeft(folks, 6, 5) == 5
* count number of sequence with combination of upper and lower case string assert(countRuns(alpa, 11) == 6); // returns 6
* count number of sequence of all upper case string assert(countRuns(alpa, 5) == 3); // return 3 A B C
* finding number of sequence from 0 element assert(countRuns(alpa, 0) == 0);
* negative n, assert(countRuns(alpa, -1) == -1); // return -1
* flip 4 element(even) assert(flip(folk, 4)==4);
* flip 3 element(odd) assert(flip(folk, 3)==3);
* flip 0 element assert(flip(folk,0) ==0);
* negative input assert(flip(folk, -1) ==-1);
* Test when there are 2 match in folkss differ(folkss, 6, groups, 5);
* Negative length, differ(folkss, -1, groups, 5);
* T1 are same up to t2 run out, and it should return the smaller n assert(differ(t1, 5, t2, 3)== 3);
* Exist one subsequence, subsequence(names, 6, names1, 3);
* No subsequence found, subsequence(names, 5, names2, 2) ;
* Exist two r2 sequences in r1, should return smallest position
* assert(subsequence(r1, 5, r2, 2)== 0);
* Subsequence if n1 is less than n2 and 0 length: subsequence(names, 0, names1, 3);
* n value is negative, subsequence(names, -1, names1, -2)
* name has one "gavin", assert(lookupAny(name, 6, set1, 4) ==1);
* a1 has none, assert(lookupAny(name, 6, set2, 2) == -1);
* n1 is negative assert(lookupAny(name, -1, set2, 2) == -1);
* n2 has 0 element,assert(lookupAny(name, 3, set2, 0) == -1);
* smallest pos of the same element in a1 assert(lookupAny(r1, 5, r2,2) == 0);
* no element in a,1 assert(lookupAny(name, 5, r2, 2) == -1);
* when divider is part of the array, assert(divide(randOrder, 9, "4")== 3);
* when divider is not part of the array divide(randOrder, 5, "4");
* when divider is the greatest of all array assert(divide(randOrder, 3, "9")==3); //n if there are no such elements
* input n is negative assert(divide(randOrder, -1, "9")==-1)