Project 4 Report

1. The first large obstacle I overcame was determining how to detect a sequence using a loop. I did not know how to stop counting matches if they were not sequential. I figured out that I can use an if statement to break the loop when it comes across a non-matching element. Therefore, the loop continues running and counting the number of matches until it reaches the first non-target element, when it will break and return “true” with the correct end value.
2. The second notable obstacle I overcame was understanding how to organize and meld together two arrays without using a third, intermediary array. Originally, I used a third array to contain all the elements of the first two, then added the minimum values of this array to the final result array. I realized that I can use the result[] array to contain all of the elements from a1[] and a2[], then do the reorganization within the result[] array through the moveToBack function and a detectMax function I made for this use.
3. My final obstacle was discovering how to deal with the function Split when the splitter element is among the elements. I needed it to somehow be in the middle of all of the other elements, but I couldn’t just transfer it into the middle without displacing other elements. I figured that I can do this using the moveToBack function, in a loop that runs as many times as there are splitters present. This will moveToBack any element that is in the center spaces that the splitters should be in-- given that the elements < splitter have already been moved to the front.

TEST DATA LIST:

[Pretend that there are quotation marks around each string within my string lists-- it was too much repetitive work to add them manually]

|  |  |
| --- | --- |
| Test Data | Reason |
| string s[6] = { “a”, “b”, ‘f”, “f”, ‘f”, “b”,};   1. countMatches(s, 5, “a”); 2. countMatches(s, 4, “f”); 3. countMatches(s, 5, “f”); 4. countMatches(s, 5, “c”); 5. countMatches(s, 6, “b”); 6. countMatches(s, -1, “a”) | 1. Checks that it returns 1 for a single match 2. Checks that it ignores matches outside of ‘n’ bounds 3. Checks that it returns the correct number for multiple sequential matches 4. Checks that it returns 0 when there is no match 5. Checks that it returns the correct number for non-sequential matches 6. Checks that it returns -1 if passed a bad argument |
| string s[6] = { “a”, “b”, “f”, “f”, ‘f”, “b”,};   1. detectMatch(s, 6, “a”); 2. detectMatch(s, 6, “b”); 3. detectMatch(s, 6, “c”); 4. detectMatch(s, -1, “f”); | 1. Checks that it returns the correct position if the match is the first element 2. Checks that it returns the smallest position if there are multiple matches 3. Checks that it returns -1 if there is no such matching string 4. Checks that it returns -1 if passed a bad argument |
| string s[11] = { “a”, “b”, “c”, ‘c”, “c, “a”, “a”, “f”, “f”, “f”, “g”,};   1. detectSequence(s, 11, “a”, b, e); 2. detectSequence(s, 11, “b”, b, e); 3. detectSequence(s, 11, “c”, b, e); 4. detectSequence(s, 11, “f”, b, e); 5. detectSequence(s, 4, “c”, b, e); 6. detectSequence(s, -1, “c”, b, e); 7. detectSequence(s, 11, “d”, b, e); | 1. Checks that it will return true if the target is the first element, and will only return b and e for the earliest occurrence 2. Checks that beginning and end will be equal if the earliest occurrence is only one string 3. Checks that b and e will be correct for a sequence of multiple matching strings 4. Checks that b and e will be correct for a sequence of multiple matching strings 5. Checks that b and e will be correct if n does not include every occurrence of the target element in its earliest sequence 6. Checks that it returns -1 if passed a bad argument 7. Checks that it returns -1 if there is no element equal to target, and b and e are unchanged |
| string s[7] = { “b”, “c”, “d”, “a”, “c”, “a”, “e”,};   1. detectMin(s, 7); 2. detectMin(s, -1); 3. detectMin(s, 0); 4. detectMin(s, 3); | 1. Checks that it returns the correct position of the smallest string (“a”), and it returns the smallest number position 2. Checks that it returns -1 if passed a bad argument 3. Checks that it returns -1 if function should examine no elements 4. Checks that it only evaluates ‘n’ interesting elements (should return 0, not 4) |
| string s[6] = { “a”, “b”, “c”, “d”, “e”, “f”, “g”};   1. moveToBack(s, 6, 1); 2. moveToBack(s, 6, 0); 3. moveToBack(s, 6, 5); 4. moveToBack(s, -1, 5); | 1. Checks that the function will eliminate the item at position pos and return position pos 2. Checks that it works if the eliminated item is in the first position 3. Checks that it works if the eliminated item is in the last position 4. Checks that it returns -1 if passed a bad argument |
| string s[6] = { “a”, “b”, “c”, “d”, “e”, “f”, “g”};   1. moveToFront(s, 6, 1); 2. moveToFront(s, 6, 0); 3. moveToFront(s, 6, 5); 4. moveToFront(s, -1, 5); | 1. Checks that the function will eliminate the item at position pos and return position pos 2. Checks that it works if the eliminated item is in the first position 3. Checks that it works if the eliminated item is in the last position 4. Checks that it returns -1 if passed a bad argument |
| string s[6] = { “a”, “b”, “c”, “d”, “e”, “f”, “g”};  string k[6] = { “a”, “b”, “d”, “d”, “e”, “f”, “g”};   1. detectDifference(s, 6, k, 6) 2. detectDifference(s, 6, k, 2) 3. detectDifference(s, 6, k, 0) 4. detectDifference(s, -1, k, 6) | 1. Checks that it returns the position of the first corresponding elements that are not equal (2) 2. Checks that it returns the value of n2 when n2 is <=n1 and the arrays are equal up until they run out 3. Checks that it returns the value of n2 when n2 = 0 < n1 4. Checks that it returns -1 if passed a bad argument |
| string s[6] = { “a”, “b”, “b”, “d’, “e”, “e”, “e”,};   1. deleteDups(s, 6) 2. deleteDups(s, 3) 3. deleteDups(s, -1)   string s[6] = { a, b, c, d, e,};   1. deleteDups(s, 6) | 1. Checks that the returned value is correct and that the identical items are removed 2. Checks that the returned value is correct and that the identical items are removed when n < the number of elements in the array 3. Checks that it returns -1 if passed a bad argument 4. Checks that it returns a number of retained elements equal to n if there are no dups |
| string little[4] = { “a”, “b”, “c”, “d”,};  string big[7] = { ‘a”, “c”, “b”, “b”, “c”, “e”, “d”,};   1. contains(big, 7, little, 4) 2. contains(big, 4, little, 4) 3. contains(big, -1, little, 4)   string little[4] = { “a”, “b”, “c”, “d”,};  string big[4] = { “a”, “d”, “c”, “b”,};   1. contains(big, 4, little, 4) 2. contains(big, 4, little, 0) | 1. Checks that it returns true when the big string contains the little string 2. Checks that it returns false when neither string contains the other 3. Checks that it returns false if passed a bad argument 4. Checks that it returns false when the strings contain the same elements in different orders 5. Checks that it returns true if one of the arrays has zero elements |
| string s[7] = { “a”, “b”, “c”, “d”, “e”, “f”, “g”, “a”,};  string k[4] = { ‘a”, “b”, “c”, “d”,};  string z[10]   1. meld(s, 6, k, 4, z, 10); 2. meld(s, 7, k, 4, z, 10); 3. meld(s, 6, k, 4, z, 5); | 1. Checks that it returns the number of combined elements of s[] and k[] when they are both in descending order and not more than max 2. Checks that it returns -1 if string a1 is not in nondecreasing order 3. Checks that it returns -1 if the result has more than max elements |
| string s[6] = { “a”, “b”, “d”, “c”, “a”, “g”, };   1. split(s, 6, “c”); 2. split(s, 6, “f”); 3. split(s, 6, “h”); 4. split(s, -1, “g”); | 1. Checks that it moves the correct elements before and after splitter when the array contains splitter 2. Checks that it moves the elements to the correct place when the array does not contain splitter 3. Checks that it returns n when there are no elements not <splitter 4. Checks that it returns -1 if the result has more than max elements |