Ashley Chang

504449890

Project 4: Arrays

One of the obstacles I faced while writing the code for this project was navigating through arrays. Because arrays act like “passing-by-reference,” I struggled with using for loops because the order of strings in the arrays kept changing, and the loop would skip some strings after another string was moved because the position was incremented. Another obstacle I faced involved using the assert function. I spent a great deal of time trying to find and fix the errors that my assert functions found, only to discover that my functions worked well, but the assert was changing the strings in my array. Thus, I discovered that I could not use multiple assert functions right after another. Yet another obstacle I faced was creating new arrays within a function. I assumed it might be easier and more effective to create new arrays to which I could concatenate strings, but I did not quite understand how I would be able to do that, as declaring arrays requires that one also declares the number of elements inside it. Finally, my last problem had to do with Visual Studio. Since I compile my programs using XCode, I had some trouble getting my last two functions (makeMerger and divide) to work on Visual Studio despite the fact that they functioned properly on XCode; whenever the main function got to them, the program would stop responding. I did not know how to fix this, and ran out of time.

List of Test Cases

|  |  |
| --- | --- |
| **Function** | **Tests** |
| Enumerate | string a[10] = {“Ashley” , “Brandon” , “Brandon” , “Brandon” , “Catherine” , “David” , “Emily” , “Floyd” , “Gabriel” , “Gabriel”};  (a, 10 , “Brandon”)   * Searches entire array * Enumerates a string that appears multiple times   (a, 5, “Catherine”)   * Does not search entire array * Enumerates a sting that only appears once   (a, 3, “Gabriel”)   * Enumerates a string that does not appear inside specified array   (a, 10, “”)   * Searches for an empty string   (a, -3, “Ashley”)   * Invalid array   (a, 0, “David”)   * Empty array |
| locate | string a[10] = {“Ashley” , “Brandon” , “Brandon” , “Brandon” , “Catherine” , “David” , “Emily” , “Floyd” , “Gabriel” , “Gabriel”};  (a, 5, “Gabriel”)   * Does not search entire array * Tries to locate string that is not in array   (a, 10, “Gabriel”)   * Searches entire array * Locates string that is in array   (a, 3, “Brandon”)   * Locates first position of string that appears more than once in indicated array   (a, 10, “Janet”)   * Searches entire array * Tries to locate string that is not existent   (a, -4, “Ashley”)   * Invalid array   (a, 0, “Ashley”)   * Empty array |
| locateSequence | string a[9] = {"jon", "daenerys", "samwell", "samwell", "margaery", "margaery", "margaery", "samwell", "samwell"};  int b;  int e;  (a, 9, “samwell”, b, e)   * Searches entire array * Locates a matching target string that appears consecutively   (a, 5, “daenerys”, b, e)   * Does not search entire array * Locates a matching target string that does not appear consecutively   (a, 9, “cersei”, b, e)   * Searches entire array * Does not locate a matching target string   (a, 4, “margaery”, b, e)   * Does not search entire array * Does not locate a matching target string (target string exists outside specified area)   (a, -4, “jon” , b, e)   * Invalid array   (a, 0, “jon”, b, e)   * Empty array |
| locationOfMin | string a[6] = {“samwell” , “jon” , “margaery” , “daenerys” , “tyrion”, “tyrion”};  (a, 6)   * Searches entire array for smallest element * Returns position of first smallest element (more than one inside array)   (a, 3)   * Searches only a portion of array for smallest element   (a, 0)   * Array has no elements, so does not find smallest element   (a, -4)   * Invalid array |
| moveToEnd | string a[5] = {“samwell” , “jon”, “margaery” , “daenerys” , “tyrion”};  (a, 3, 3)   * Item at pos is outside of range * Does not use entire array   (a, 5, 4)   * Uses entire array * Item at pos is already at end; position does not change   (a, 4, 2)   * Does not use entire array * Item at pos is moved to end of indicated sub-array   (a, 5, 1)   * Uses entire array * Item at pos is moved to end   (a, -5, 4)   * Invalid array   (a, 0, 3)   * Empty array |
| moveToBeginning | string a[5] = {“samwell” , “jon”, “margaery” , “daenerys” , “tyrion”};  (a, 5, 5)   * Item at pos is outside of range * Uses entire array   (a, 4, 0)   * Does not use entire array * Item at pos is already at beginning; position does not change   (a, 3, 1)   * Does not use entire array * Item at pos is moved to beginning of indicated subarray   (a, 5, 3)   * Uses entire array * Item at pos is moved to beginning   (a, -5, 2)   * Uses invalid array   (a, 0, 3)   * Empty array |
| locateDifference | string a[5] = { "apple", "banana", "carrot", "pickle", "jellyfish" };  string b[4] = { "apple", "banana", "dog", "blueberry"};  (a, 5, b, 4)   * Uses entirety of both arrays * Difference occurs before either array ends   (a, 3, b, 2)   * Does not use the entirety of either array * One array runs out before the other * Arrays are the same up until second array runs out   (a, 3, b, 3)   * Arrays are the same size * Difference occurs before either array ends * Does not use the entirety of either array   (a, 0, b, 4)   * One array has no elements, other uses all elements   (a, 2, b, 2)   * Exactly the same array (subarrays); no differences to be found   (a, -3, b, 5)   * Contains an invalid array |
| eliminateDups | string a[9] = {"jon", "daenerys", "samwell", "samwell", "margaery", "margaery", "margaery", "samwell", "samwell"};  (a, 9)   * Uses entire array * Finds and removes duplicates (retains single string)   (a, 3)   * Does not use entire array * No duplicates to remove   (a, 6)   * Does not use entire array * Finds and removes duplicates   (a, -3)   * Uses invalid array   (a, 0)   * One of arrays is empty |
| subsequence | string a[8] = {“apple”, “banana”, “carrot”, “pickle”, “unicorn”, “cake”, “lollipop”, “lemon”};  string b[3] = {“apple” , “unicorn”, “lemon”};  (a, 8, b, 3)   * Uses entirety of both arrays * All elements in small array appear in large array in same order   (a, 1, b, 2)   * Second array is larger than first * Does not use entirety of either array   (a, 7, b, 3)   * Not all elements of second array appear in first array   (a, 7, b, 2)   * All elements in small array appear in large array in same order * Does not use entirety of either array   (a, 4, b, 0)   * One of arrays is empty   string c[3] = { “”, “eeyore”, “pooh” };  (a, 8, c, 3)   * None of elements of second array appear in first array   string d[4] = {“carrot” , “apple”, “lollipop”}  (a, 8 , d, 4)   * Elements in small array are subarray of large array, but not all in order |
| makeMerger  \*\*\*Function did not work in Visual Studio | string a[5] = { "cersei", "jon", "margaery", "samwell", "sansa" };  string b[4] = { "daenerys", "jon", "jon", "tyrion" };  string z[20];  (a, 5, b, 4, z, 10)   * Uses entirety of both arrays * Both arrays in nondecreasing order * Result does not exceed max   (a, 3, b, 3, z, 10)   * Does not use entirety of either array * Both arrays in non decreasing order * Result does not exceed max   (a, 0, b, 3, z, 10)   * One of arrays is empty   (a, 4, b, 4, z, 6)   * Requires more than max elements   (a, -2, b, 3, z, 10)   * Invalid array   (a, 3, b, 5, z, -10)   * Invalid max   string c[3] = {“zoobamafoo”, “lion”, “panda”}  (a, 5, c, 3, z, 10)   * One array is not in nondecreasing order |
| Divide  \*\*\*Function did not work in Visual Studio | string a[6] = {“apple”, “orange”, “banana”, “banana”, “grape”, “pear”}  (a, 5, “carrots”)   * Does not use entire array * Uses divide string that is not already in array   (a, 6, “banana”)   * Uses divide string that is already in array * Uses entire array   (a, 4, “apple”)   * No elements before divide   (a, 6, “pear”)   * No elements after divide   (a, -1, “grape”)   * Invalid array   (a, 0, “orange”)   * Empty array |