Name: Joseph Picchi

UID: 605-124-511

Project 4 Report

a) Notable obstacles I overcame include:

* Learning how to loop through a string when that string in itself is an element of an array. I ended up doing so with two indexes:

array[4][5]

The index 4 represents the position of the string in the array, and the index 5 denotes the position of the desired character in the string.

* Learning how to test many different input arrays at once, which I discovered is a job best left to the assert macro. I didn’t use assert statements in project 4, and I instead would input each string manually using a getline statement, but I found that using assert statements in this project made my work much more efficient and made my code much easier to test.
* Understanding the nature of the array parameter. Through experimentation, I discovered that the array parameter behaves very similarly to a pass-by-reference parameter in that changes made to the array by a function can be seen outside the scope of that function. In the same manner, I also came to realize that the const modifier should be added to make sure a function does not change an array.

b) To test my functions, I first initialized these strings:

string folks[8] = {“samwell”, “jon”, “margaery”, “daenerys”, “tyrion”, “sansa”, “howard123”, “jon”};

string allLetters[4] = {“math”, “science”, “bread”, “water”};

string largeSized[21] = {“water”, “bread”, “cheese”, “fig”, “drink”};

string isDecreasingOrder[5] = {“Elephant”, “Dog”, “Cat”, “Betafish”, “Ant”};

string shiftTest[6] = {“foo”, “zeta”, “foo”, “cat”, “dog”};

string findTest[6] = {“foo”, “test”, “foo”, “ “, “”, “test”};

The following test data could be used to thoroughly test my functions:

*Testing the hasDuplicates function:*

|  |  |
| --- | --- |
| **Test Data** | **Reason for Test** |
| assert(!hasDuplicates(folks, 5)); | Should return false for an array with no duplicate entries. |
| assert(hasDuplicates(folks, 8)); | Should return true for an array with duplicate entries. |
| assert(!hasDuplicates(folks, -1)); | Should return false for a size less than zero. |
| assert(!hasDuplicates(folks, 0)); | Should return false for a size equal to zero. |
| assert(!hasDuplicates(largeSized, 21)); | Should return true for an extremely large array that has duplicate empty strings as entries. |

*Testing the countAllDigits function:*

|  |  |
| --- | --- |
| **Test Data** | **Reason for Test** |
| assert(countAllDigits(folks, 8) == 3); | Should return the number of digit characters ‘0’ - ‘9’ that appear in an array. |
| assert(countAllDigits(folks, -3) == -1); | Should return -1 for a size less than zero. |
| assert(countAllDigits(folks, 0) == -1); | Should return -1 for a size of zero. |
| assert(countAllDigits(allLetters, 4) == 0); | Should return 0 for an array with no digits |

*Testing the isInDecreasingOrder function:*

|  |  |
| --- | --- |
| **Test Data** | **Reason for Test** |
| assert(isInDecreasingOrder(isDecreasingOrder, 5)); | should return true if every value in the array is smaller than the one that precedes it. |
| assert(isInDecreasingOrder(isDecreasingOrder, 3)); | Should return true if an array is in decreasing order and the inputed size is less than that of the array. |
| assert(!isInDecreasingOrder(folks, 8)); | Should return false for an array that is not in decreasing order. |
| assert(!isInDecreasingOrder(folks, -3)); | Should return false for a size less than zero. |
| assert(isInDecreasingOrder(folks, 0)); | Should return true for a size equal to 0. |
| assert(!isInDecreasingOrder(folks, 1)); | Should return true for a size equal to 1. |

*Testing the shiftRight function:*

|  |  |
| --- | --- |
| **Test Data** | **Reason for Test** |
| assert(shiftRight(folks, -2, 3, “hello) == -1); | should return -1 if the size is less than 0. |
| assert(shiftRight(folks, 2, -3, “hello) == -1); | should return -1 for an amount less than 0. |
| assert(shiftRight(folks, 8, 3, “hello) == 3);  assert(folks[0] == “hello”);  assert(folks[1] == “hello”);  assert(folks[2] == “hello”);  assert(folks[3] == “samwell”);  assert(folks[4] == “jon”);  assert(folks[5] == “margaery”);  assert(folks[6] == “daenerys”);  assert(folks[7] == “tyrion”); | Should return 3 because that is the number of times the placeholder parameter was added. Each element in the array should also be shifted to the right as indicated. |
| assert(shiftRight(shiftTest, 6, 4, “foo”) == 4); | Should return the amount of times the placeholder “foo” was added, not the number of times it appears in the resulting string. |

*Testing the find function:*

|  |  |
| --- | --- |
| **Test Data** | **Reason for Test** |
| assert(find(findTest, 6, “foo”) == 0); | should return the smallest index of the element that matches the parameter “foo”. |
| assert(find(findTest, -1, “foo”) == -1); | should return -1 if the size is less than zero. |
| assert(find(findTest, 0, “foo”) == 0); | should return -1 if the size is equal to zero. |
| assert(find(findTest, 6, “dog”) == -1); | should return -1 if the element is not found. |

*Testing the replaceAllCharacters function:*

|  |  |
| --- | --- |
| **Test Data** | **Reason for Test** |
| assert(replaceAllCharacters(folks, 0, ‘a’, ‘\_’) == -1); | should return -1 for a size of zero. |
| assert(replaceAllCharacters(folks, -25, ‘a’, ‘\_’) == -1); | should return -1 for a size less than zero. |
| assert(replaceAllCharacters(folks, 8, ‘3’, ‘\_’) == 0);  assert(folks[2] == “margaery”); | should return 0 if the parameter is not found in the array, and all elements of the array should remain unchanged. |
| assert(replaceAllCharacters(folks, 5, ‘a’, ‘\_’) == 4);  assert(folks[2] == “m\_arg\_ery”); | should return the number of times the parameter is found in the string, and the character should be replaced with the replacement parameter in each element of the array. |

The program I created passes all of these tests.