1. A brief description of notable obstacles you overcame.

For my ReplaceFirstandLastOccurances function I accidentally wrote a for statement which looked at the array at the position array.size() at the beginning which caused an out of range error. Furthermore, I was comparing a Array\_i.size() to n which was an error because I was comparing a size\_t to an int. So, I had to do a static\_cast in order to fix my error. After that, I also had to change my ShiftLeft function. I was decrementing the argument “amount” whereas I should have decremented a copy of that variable in order to return “amount” at the end.

1. 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.

Note: for the sake of saving my own time, I will write the array out inside the function whereas I would have to declare and initialize the array and then use the array name inside the function in a normal program.

General Test Case:

My functions fail if the array size n given by the user is bigger than the actual array size. My function assumes that the number n given by the user makes sense.

**FUNCTION: locateMaximum(const string array{ }, int n )**

**Reason: must return -1 if n <= 0**

Input 1: locateMaximum({ “” } , -8989 ) ====> -1

Input 2: locateMaximum({“” , “hello” } , 0) ====> -1

**Reason:** **if there are duplicate maximum values, must return the smallest index that has the maximum value**

Input 1: locateMaximum({ “” , “Z” , d” , “” , “Z” , “Z”, “d”}, 4) ====> 2

Input 2: locateMaximum({ “” , “Z” , d” , “” , “Z” , “Z”, “d”}, 7) ====> 2

**Reason: must sort correctly when there are array elements that are words or symbols**

Input 3: locateMaximum({ “ASDIDAIDFUIOD” , “HSAJFHJK” , “Z” , “XJKSHD” , “” , “XOPOO” , “d”}, 7) ====> 6

Input 4: locateMaximum({ “ASDIDAIDFUIOD” , “HSAJFHJK” , “!” , “XJKSHD” , “” , “XOPOO” , “d”}, 6) ====> 5

Input 5: locateMaximum({ “!” , “ .” , “\\” , “#” }, 4) ====> 2

**FUNCTION: countFloatingPointValues(const string array{ }, int n )**

**Reason: must return -1 if n <= 0**

Input 1: countFloatingPointValues({“1”, “2”, “3”, “4”}, -79) ====> -1

Input 2: countFloatingPointValues({“1”, “2”, “3”, “4”}, 0) ====> -1

**Reason: must include at least 1 number and only include numbers or a single optional decimal point (cannot have +, -, more than one decimal point, letters, symbols or ,)**

Input 1: countFloatingPointValues({“+1”, “-2”, “300,000” , “#1” , “567!” , “100.000.000” , “10..” , “AA” , “100.0”}, 9) ====> 1

Input 2: countFloatingPointValues({“+1”, “-2”, “300,000” , “#1” , “567!” , “100.000.000” , “10..” , “AA” , “100.0”}, 8) ====> 0

Input 3: countFloatingPointValues({“.” , “0000123231”, “1234” ,“1232.23000”}, 4} ====> 3

Input 4: countFloatingPointValues({“”}, 1) ====> 0

Input 5: countFloatingPointValues({ "A1", ".", "A", "123.3d", ""}, 5) ====> 0

**FUNCTION:** **hasNoCapitals(const string array{ }, int n )**

**Reason: must return true if n <= 0**

Input 1: hasNoCapitals({ "A", "B", "C", "B" }, -4040) ====> true

Input 2: hasNoCapitals({ "A", "B", "C", "B" }, 0) ====> true

**Reason: must return false if there are capital letters and true if there are no capital letters**

Input 1: hasNoCapitals({ "A", "B", "C", "B" }, 4 ) ====> false

Input 2: hasNoCapitals({ "a", "b", "c", "d" }, 4 ) ====> true

Input 3: hasNoCapitals({ “ adshfjhjK”, “a”}, 2) ====> false

Input 4: hasNoCapitals({ “ adshfKjhj”, “a”}, 2) ====> false

Input 5: hasNoCapitals({ “ adshfjhj”, “#!\\”}, 2) ====> true

Input 6: hasNoCapitals({“”}, 1) ====> true

**FUNCTION: identicalValuesTogether(const string array[ ], int n)**

**Reason: must return false if n <= 0**

Input 1: hasNoCapitals({ "B", "B", "C", "C" }, -4040) ====> false

Input 2: hasNoCapitals({ "B", "B", "C", "C" }, 0) ====> false

**Reason: all identical values must be adjecant to at least one other identical value if there are at least two of the same string in the array.**

Input 1: identicalValuesTogether({ "A", "B", "C", "B" }, 4 ) ====> false

Input 2: identicalValuesTogether({ "a", "b", "b", "d" }, 4 ) ====> true

Input 3: identicalValuesTogether({ "a", "a", "b", "c", "b" }, 5 ) ====> false

Input 4: identicalValuesTogether({ "a", "b", "c", "d", "e" }, 5 ) ====> true

Input 5: identicalValuesTogether({ "a", "a", "b", "a", "a" }, 5 ) ====> false

Input 6: identicalValuesTogether({ " ", " ", "#", "a", "a" }, 5 ) ====> true

Input 7: identicalValuesTogether({ "asdfghjkl ", "asdfghjk ", "asdfghjkl", "a", "a" }, 5 ) ====> false

Input 8: identicalValuesTogether({ "1234", "1234", "asdfghjkl", "#", "#" }, 5 ) ====> true

**FUNCTION: hasTwoOrMoreDuplicates(const string array[ ], int n )**

**Reason: must return false if n <= 0**

Input 1: hasTwoOrMoreDuplicates({ "A", "B", "C", "B" }, -4040) ====> false

Input 2: hasTwoOrMoreDuplicates({ "A", "B", "C", "B" }, 0) ====> false

**Reason: must return true if there are three array elements which contain the same string or if there are at least two pairs (a pair is made up of two array elements which contain the same string) in the array.**

**Note: the array elements of different pairs can contain different strings, but the array elements of the same pair must contain the same string.**

Input 1: hasTwoOrMoreDuplicates({ "A", "B", "C", "B" }, 4 ) ====> false

Input 2: hasTwoOrMoreDuplicates({ "a", "a", "b", "b" }, 4 ) ====> true

Input 3: hasTwoOrMoreDuplicates({ "a", "a", "a", "c", "b" }, 5 ) ====> true

Input 4: hasTwoOrMoreDuplicates({ "a", "a", "a", "a", "b" }, 5 ) ====> true

Input 5: hasTwoOrMoreDuplicates({ "a", "b", "a", "b", "a" }, 5 ) ====> true

Input 6: hasTwoOrMoreDuplicates({ "asdfghjkl", "b#", "asdfghjkl", "b", "a" }, 4 ) ====> false

Input 7: hasTwoOrMoreDuplicates({ "", "", "", "", "" }, 5 ) ====> true

Input 8: hasTwoOrMoreDuplicates({ "123", "", "23", "", "23"}, 5 ) ====> true

Input 9: hasTwoOrMoreDuplicates({ "123", "", "23", "#", “#” , “!”, “!”}, 7 ) ====> true

Input 10: hasTwoOrMoreDuplicates({ "B", "B", "C", "Basdsdafsf" }, 4 ) ====> false

**FUNCTION:** **shiftLeft(string array[ ], int n, int amount, string placeholder)**

**Reason: must return -1 if n <= 0**

Input 1:shiftLeft({ "A", "B", "C", "B" }, -4040, 12, "foo") ====> -1

Input 2:shiftLeft({ "A", "B", "C", "B" }, 0, 12, "foo") ====> -1

**Reason: must return -1 if amount < 0**

shiftLeft({ "A", "B", "C", "B" }, 4, -12, "foo" ) ====> -1

**Reason: if amount => n, then the array should be shifted n times and return n**

Input 1: shiftLeft({ "a", "a", "a", "c", "b" }, 5, 20, "foo" ) ====> 5, {"foo", "foo", "foo", "foo", "foo"}

Input 2: shiftLeft({ "a", "a", "a", "c", "b" , “b”, “b”}, 5, 20, "foo" ) ====> 5, {"foo", "foo", "foo", "foo", "foo", “b”, “b”}

Input 3: shiftLeft({ "a", "a", "a", "c", "b" }, 5, 5, "foo" ) ====> 5, {"foo", "foo", "foo", "foo", "foo"}

**Reason: the array must be altered and shifted “amount” times to the left**

Input 1: shiftLeft({ "a", "a", "foo", "c", "b" }, 5, 1, "foo" ) ====> 1, {"a", "foo", "c", "b", "foo"}

Input 2: shiftLeft({ "A", "B", "C", "B" }, 4, 0, "foo") ====> 0, {"A", "B", "C", "B"}

Input 3: shiftLeft({ "a", "a", "a", "c", "b" }, 5, 2, "foo" ) ====> 2, {"a", "c", "b", "foo", "foo"}

Input 4: shiftLeft({ "a", "a", "foo", "c", "b" }, 5, 1, "foo" ) ====> 1 , {"a", "foo", "c", "b", "foo"}

Input 5: shiftLeft({ "asdfghj", "a", "foo", "c", "bsdfghj" }, 5, 1, "foo" ) ====> 1 , {"a", "foo", "c", "bsdfghj", "foo"}

**FUNCTION:** **replaceFirstAndLastOccurrences( string array[ ], int n, char charToFind, char charToReplace )**

**Reason: must return -1 if n <= 0**

Input 1: replaceFirstAndLastOccurrences( "A", "B", "C", "B" }, -4, 'A', 'z' ) ====> -1

Input 2: replaceFirstAndLastOccurrences({ "A", "B", "C", "B" }, 0, 'A', 'z' ) ====> -1

**Reason: must return 0 if no charToFind is found**

Input 1: replaceFirstAndLastOccurrences( { "A", "B", "C", "B" }, 4, 'P', 'z' ) ====> 0

**Reason: must not erroneously replace a charToFind twice**

Input 1: replaceFirstAndLastOccurrences({ "A", "B", "C", "B" }, 4, 'A', 'z' ) ====> 1 , { "z", "B", "C", "B" }

**Reason: must only replace the first and last charToFind per string**

Input 1: replaceFirstAndLastOccurrences( { "AAA", "ABA", "ACAAA", "BA" }, 4, 'A', 'z') ====> 7 ,

{"zAz", "zBz", "zCAAz", "Bz"}

Input 2: replaceFirstAndLastOccurrences({ "AAA", "ABA", "ACAAA", "zzzzzA" }, 4, 'A', 'z') ====> 7 ,

{"zAz", "zBz", "zCAAz", "zzzzzz" }

Input 3: replaceFirstAndLastOccurrences( { "AAA", "ABA", "ACAAA", "zzzzzA" }, 4, 'A', 'A') ====> 7 , {"AAA", "ABA", "ACAAA", "zzzzzA" }

Input 4: replaceFirstAndLastOccurrences( { "AAA", "ABA", "ACAAA", "zzzzzA" }, 4, 'A', ' ') ====> 7 , {" A ", " B ", " CAA ", "zzzzz " }