Testing Report

Testing of method1

- Test 0, 1 and many: These tests will check the alphabetical order for the strings having length 0, 1 and more than 1 i.e having one 0 character, 1 character and more than one character stored in the string
- Test first, last and middle: They don't make any sense as this method doesn't searches or modifies the string

Test 0, Test 1 and Test many

Test 0

Giving the string length of the input string in the method = 0

In the interaction, typed the following:

HW2.isAlphabeticalOrder("")

Expected Output: true

Actual Output: true

Test 1

Giving the string length of the input string in the method = 1

In the interaction, typed the following:

> HW2.isAlphabeticalOrder("H")

Expected Output: true (Since, there is only one character in the string, it is pointless to say whether the string is in alphabetical order or not. So by default it returns true)

Actual Output: true

Test many

Giving the string length of the input string in the method = more than one (5)

In the interaction, typed the following:

HW2.isAlphabeticalOrder("H!iop")

Expected Output: true (Since, all the alphabets are in the order irrespective of their cases and the position of other characters, the output should be true)

Actual Output: true

Since the Test 0, 1 and many are successful; the method 1 works perfectly!

Testing of method2

- Test 0, 1 and many: These tests will check the ability of method to remove the given character 0 times, 1 time and more than 1 time from the given string for the strings having length 0, 1 and more than 1 and
- Test first, last and middle: These tests will find the character to be removed present at first, last or in the middle of the string

Let the number of the characters to be removed = n

Let the string length of the string = l

Test 0, Test 1 and Test many

When l=0, n=0

In the interaction, typed the following:

> HW2.removeNchars("",0,'c')

Expected Output :""

Actual Output : ""

When l=1, n=0

When character to be removed was present in the string

In the interaction, typed the following:

> HW2.removeNchars("c",0,'c')

Expected Output: "c"

Actual Output: "c"

When character to be removed wasn't present in the string

In the interaction, typed the following:

> HW2.removeNchars("v",0,'c')

Expected Output: "v"

Actual Output: "v"

When l=1, n=1

When character to be removed was present in the string

In the interaction, typed the following:

> HW2.removeNchars("c",1,'c')

Expected Output: ""

Actual Output: ""

When character to be removed wasn't present in the string

> HW2.removeNchars("v",1,'c')

Expected Output: "v"

Actual Output: "v"

When l=7, n=0

In the interaction, typed the following:

> HW2.removeNchars("vcgitpc",0,'c')

Expected Output: "vcgitpc"

Actual Output: "vcgitpc"

When l=7, n=1

In the interaction, typed the following:

> HW2.removeNchars("vcgitpc",1,'c')

Expected Output: "vgitpc"

Actual Output: "vgitpc"

When l=7, n=2

In the interaction, typed the following:

> HW2.removeNchars("vcgitpc",2,'c')

Expected Output: "vgitp"

Actual Output: "vgitp"

Test first, last and middle

Test First

When the character to be removed was the first character in the string

In the interaction, typed the following:

> HW2.removeNchars("cvgitp",1,'c')

Expected Output: "vgitp"

Actual Output: "vgitp"

Test Last

When the character to be removed was at the last character in the string

In the interaction, typed the following:

> HW2.removeNchars("vgitpc",1,'c')

Expected Output: "vgitp"

Actual Output: "vgitp"

Test Middle

When the character to be removed was in the middle of the string

In the interaction, typed the following:

> HW2.removeNchars("vgcitp",1,'c')

Expected Output: "vgitp"

Actual Output: "vgitp"

Since the Test 0, 1 and many and Test first, last and middle are successful; the method 2 works perfectly!

Testing of method 3

- Test 0, 1 and many: These tests will check the ability of the method to remove the given smaller string with length 0, 1, and more than 1 from the given larger string having length 0, 1 and more than 1
- Test first, last and middle: These tests will check the ability to search the shorter string to be removed, present at first, last or in the middle part of the larger string

Let the length of the smaller string = l

Let the length of the larger string = L

Test 0, 1 and many

When l=0, L=0

In the interaction, typed the following:

> HW2.removeString("","")

Expected Output: ""

Actual Output: ""

When l=0, L=1

In the interaction, typed the following:

> HW2.removeString("a","")

Expected Output: "a"

Actual Output: "a"

When l=0, L=5

In the interaction, typed the following:

> HW2.removeString("abcde","")

Expected Output: "abcde"

Actual Output: "abcde"

When l=1, L=0

In the interaction, typed the following:

> HW2.removeString("","a")

Expected Output: ""

Actual Output: ""

When l=1, L=1

In the interaction, typed the following:

> HW2.removeString("a","a")

Expected Output: ""

Actual Output: ""

When l=1, L=5

In the interaction, typed the following:

> HW2.removeString("abcde","a")

Expected Output: "bcde"

Actual Output: "bcde"

When l=5, L=0

In the interaction, typed the following:

> HW2.removeString("","abcde")

Expected Output: ""

Actual Output: ""

When l=5, L=1

In the interaction, typed the following:

> HW2.removeString("a","abcde")

Expected Output: "a"

Actual Output: "a"

When l=5, L=5

In the interaction, typed the following:

> HW2.removeString("abcde","abcde")

Expected Output: ""

Actual Output: ""

Test First, Last and Middle

Test First

When the string to be removed was in the first part of the string

In the interaction, typed the following:

> HW2.removeNchars("abcdef","abc")

Expected Output: "def"

Actual Output: "def"

Test Last

When the string to be removed was in the last part of the string

In the interaction, typed the following:

> HW2.removeNchars("abcdef","def")

Expected Output: "abc"

Actual Output: "abc"

Test Middle

When the string to be removed was in the Middle part of the string

In the interaction, typed the following:

> HW2.removeNchars("abcdef","cd")

Expected Output: "abef"

Actual Output: "abef"

Since the Test 0, 1 and many and Test first, last and middle are successful; the method 3 works perfectly!

Testing of method 4

- Test 0, 1 and many: These tests will check the ability of the method to shifts the character to right when in a string having length 0, 1, and more than 1
- Test first, last and middle: These tests will check the ability to search the character in the string and shift it right present at first, last or in the middle part of the string

Test 0, 1 and many

When length of the string = 0

In the interaction, typed the following:

>HW2.moveAllXsRight('a',"")

Expected Output: ""

Actual Output: ""

When length of the string = 1

In the interaction, typed the following:

> HW2.moveAllXsRight('a',"a")

Expected Output: "a"

Actual Output: "a"

When length of the string = 7

In the interaction, typed the following:

> HW2.moveAllXsRight('a',"abcdeav")

Expected Output: "bacdeva"

Actual Output: "bacdeva"

Test First

When the character to be moved is the first character of the string

In the interaction, typed the following:

HW2.moveAllXsRight('a',"abcdev")

Expected Output: "bacdev"

Actual Output: "bacdev"

Test Last

When the character to be moved is the last character of the string

In the interaction, typed the following:

> HW2.moveAllXsRight('a',"bcdeva")

Expected Output: "bcdeva"

Actual Output: "bcdeva"

Test Middle

When the character to be moved is the Middle character of the string

In the interaction, typed the following:

> HW2.moveAllXsRight('a',"bcdaev")

Expected Output: "bcdeav"

Actual Output: "bcdeav"

Since the Test 0, 1 and many and Test first, last and middle are successful; the method 4 works perfectly!

Testing of method 5

- Test 0, 1 and many: These tests will check the ability of the method to shift the every occurence of the given character in 2-D array to the bottom most possible when the arrays in the 2D array have length 0, 1, and more than 1
- Test first, last and middle: These tests will check the ability of the method to shift the every occurence of the given character in 2-D array to the bottom most possible when the first character is present in first array, last array and in the middle array of the 2D array

Test 0,1 and many

Test 0

When there is no character in the 2-D array

In the interaction, typed the following:

> char array[][] = {{' '},{' '},{' '}}

> HW2.moveAllXsdown('a',array)

> array

Expected Output: { { }, { }, { } }

Actual Output: { { }, { }, { } }

Test 1

When there is 1 character in the 2-D array

In the interaction, typed the following:

> char array[][] = {{'b'},{'a'},{'c'}}

> HW2.moveAllXsdown('a',array)

> array

Expected Output: { { b }, { c }, { a } }

Actual Output: { { b }, { c }, { a } }

Test Many

When there is more than one character in the 2-D array

In the interaction, typed the following:

> char array[][] = {{'a','b','c','X'},{'d','X','e','f','X'},{'X','X','i'},{'X','j','k','l'}};

> HW2.moveAllXsdown('X',array)

> array

Expected Output: { { a, b, c, f }, { d, j, e, X, X }, { X, X, i }, { X, X, k, l } }

Actual Output: { { a, b, c, f }, { d, j, e, X, X }, { X, X, i }, { X, X, k, l } }

Test First, Last and Middle

Test First

When there is character in the first array of 2-D array

In the interaction, typed the following:

> char array[][] = {{'a','a','c'},{'g','e','o'},{'a','e','t', 'r'}}

> HW2.moveAllXsdown('a',array)

> array

Expected Output: { { g, e, c }, { a, e, o }, { a, a, t, r } }

Actual Output: { { g, e, c }, { a, e, o }, { a, a, t, r } }

Test Last

When there is character in the last array of 2-D array

In the interaction, typed the following:

> char array[][] = {{'b','t','c','w'},{'g','e','t'},{'a','e','t','r','a'}}

> HW2.moveAllXsdown('a',array)

> array

Expected Output: { { b, t, c, w }, { g, e, t }, { a, e, t, r, a } }

Actual Output: { { b, t, c, w }, { g, e, t }, { a, e, t, r, a } }

Test Middle

When there is character in the middle array of 2-D array

In the interaction, typed the following:

char array[][] = {{'b','t','c','w'},{'g','a','a'},{'h','e','t','r','q'}}

> HW2.moveAllXsdown('a',array)

> array

Expected Output: { { b, t, c, w }, { g, e, t }, { h, a, a, r, q } }

Actual Output: { { b, t, c, w }, { g, e, t }, { h, a, a, r, q } }

Since the Test 0, 1 and many and Test first, last and middle are successful; the method 5 works perfectly!

Testing of method 6

- Test 0, 1 and many: These tests will check the ability of the method to shift every occurrence of the given character in 2-D array to the bottom most possible when the arrays in the 2D array have length 0, 1, and more than 1
- Test first, last and middle: These tests will check the ability of the method to shift every occurrence of the given character in 2-D array to the bottom most possible when the first character is present in first array, last array and in the middle array of the 2D array

Test 0,1 and many

Test 0

When there is no character in the 2-D array

In the interaction, typed the following:

char [][] board = {{' '},{' '},{' '}}

> HW2.moveXDownLeft('X',board)

> board

Expected Output: { { }, { }, { } }

Actual Output: { { }, { }, { } }

In the interaction, typed the following:

char [][] board = {{'a'},{'b'},{'c'}}

> HW2.moveXDownLeft('X',board)

> board

Expected Output: { { a }, { b }, { c } }

Actual Output: { { a }, { b }, { c } }

Test 1

When there is only one character in the 2-D array

In the interaction, typed the following:

> char [][] board = {{'a'},{'X'},{'f',}}

> HW2.moveXDownLeft('X',board)

> board

Expected Output: { { a }, { X }, { f, } }

Actual Output: { { a }, { X }, { f, } }

Test many

When there is more than character in the 2-D array

In the interaction, typed the following:

char[][] board = {{'a','b','c','X'},{'d'},{'e','f','g','h'},{'i','j','k'},{'l','m','n','o'}};

> HW2.moveXDownLeft('X',board)

> board

Expected Output: { { a, b, c, f }, { d }, { e, i, g, h }, { X, j, k }, { l, m, n, o } }

Actual Output: { { a, b, c, f }, { d }, { e, i, g, h }, { X, j, k }, { l, m, n, o } }

Test First, Last and Middle

Test First

When there is character in the first arrayof 2-D array

In the interaction, typed the following:

```
char[][] board = {{'a','b','c','X'},{'d'},{'e','f','g','h'},{'i','j','k'},{'l','m','n','o'}};
> HW2.moveXDownLeft('X',board)
> board
```

Expected Output: { { a, b, c, f }, { d }, { e, X, g, h }, { i, j, k }, { l, m, n, o } }

Actual Output: { { a, b, c, f }, { d }, { e, X, g, h }, { i, j, k }, { l, m, n, o } }

Test Middle

When there is character in the Middle array of 2-D array

In the interaction, typed the following:

```
> char[][] board = {{'a','b','c','q'},{'d'},{'e','f','X','h'},{'i','j','k'},{'l','m','n','o'}};
> HW2.moveXDownLeft('X',board)
> board
```

Expected Output: { { a, b, c, q }, { d }, { e, f, j, h }, { i, X, k }, { l, m, n, o } }

Actual Output: { { a, b, c, q }, { d }, { e, f, j, h }, { i, X, k }, { l, m, n, o } }

Test Last

When there is character in the last array of 2-D array

In the interaction, typed the following:

```
char[][] board = {{'a','b','c','q'},{'d'},{'e','f','g','h'},{'i','j','k'},{'l','X','n','o'}};
> HW2.moveXDownLeft('X',board)
> board
```

Expected Output: { { a, b, c, q }, { d }, { e, f, g, h }, { i, j, k }, { l, X, n, o } }

Actual Output: { { a, b, c, q }, { d }, { e, f, g, h }, { i, j, k }, { l, X, n, o } }

Since the Test 0, 1 and many and Test first, last and middle are successful; the method 5 works perfectly!

Testing of method 7

Test 0, 1 and many: These tests will check the ability of the method to shift every occurrence of the given character in the string to the bottom most possible when the string has length 0, 1, and more than 1

Test first, last and middle: These tests will check the ability of the method to shift every occurrence of the given character in the string to the bottom and right most possible when the character to be shifted is present in first index, last index and in the middle index of the string

The method is not completely perfect. It is not giving the perfect output

HW2.moveXDownRight('X', "Xabc\nd\nefgh\nijk\nlmnop")

Expected Output: "gabc

d

efph

ijk

lmnoX"

Actual Output: "gabc

d

efpgh

ijk

lmnoX"

So, had the method been working perfectly, I would have done the following tests (as described below)

Test 0,1 and many

Test 0

When the length of the string =0

In the interaction, type the following:

HW2.moveXDownRight('X', "")

When the length of the string more than 1 and the character is not present in the string

In the interaction, type the following:

HW2.moveXDownRight('X', "abc\nd\nefgh\nijk\nlmnop")

Test 1

Possible cases

When the length of the string = 1 and the input character is the character

In the interaction, type the following:

>HW2.moveXDownRight('X', "X")

Expected Output: “X”

When the length of the string = 1 and the input character isn’t present in the string

In the interaction, type the following:

>HW2.moveXDownRight('X', "p")

Expected Output: “X”

Test Many

Possible cases

When length of the string is more than 1 and the input character is not present in the string

In the interaction, type the following:

>HW2.moveXDownRight('X', "Sabc\nd\nefgh\nijk\nlmnop")

Expected Output:

"Sabc

d

efgh

ijk

lmnop"

When length of the string is more than 1 and the input character is present more than once

In the interaction, type the following:

HW2.moveXDownRight('X', "XabX\nd\nefgh\nijk\nlmnop")

Expected Output:

"gabX

d

efph

ijk

lmnoX"

Test First, Last and Middle

Test First

When the length of the string is more than 1 and the input character is the first character of the string

In the interaction, type the following:

HW2.moveXDownRight('X', "Xabc\nd\nefgh\nijk\nlmnop")

Expected Output:

"gabc

d

efph

ijk

lmnoX"

Test Last

When the length of the string is more than 1 and the input character is the last character of the string

In the interaction, type the following:

HW2.moveXDownRight('X', "Sabc\nd\nefgh\nijk\nlmnoX")

Expected Output:

"Sabc

d

efgh

ijk

lmnoX"

Test Middle

When the length of the string is more than 1 and the input character is in the mid of the string

In the interaction, type the following:

HW2.moveXDownRight('X', "Sabc\nd\nefXh\nijk\nlmnop")

Expected Output:

"Sabc

d

efh

ijkp

lmnoX"