Homework 4

One way to represent a sequence of characters is to use a `String`. Another way is to use an array of `char` values (which, from now on, we call "char arrays").

Internally, strings are handled as char arrays. For example, when you say `String s = "IDC"`, the Java compiler generates low-level code that implements the operations `char[] s = new char[3]; s[0] = 'I'; s[1] = 'D'; s[2] = 'C';`.

Likewise, all the string methods, like `s.chartAt` and `s.indexOf`, are implemented internally as operations on the equivalent char array `s`. Here are some examples of this equivalence:

Abstraction (what the Java programmer writes / reads)	**Implementation** (what the code generated by the compiler does)
`String s = "Hello World"`	`char[] s = new char[11];` `s[0] = 'H'; s[1] = 'e'; ... s[11] = 'd';`
`s.length()`	Returns `s.length` // The number of elements in the array
`s.toString()`	Returns a string consisting of all the elements of `s`, one after the other
`s.charAt(k)`	`s[k]`
`s1 + s2`	Returns a new char array whose length is the sum of lengths of the char arrays that represent `s1` and `s2`, then populates the new array with the elements of `s1` and `s2`.
Etc.	Every operation on the string `s` is implemented as an operation on the array of char values that represents s.

The `CharArray` class

This class implements various String functions as functions that operate on char arrays. Your task is to complete the implementation of the following functions:

`public static String toString(char[] arr)` (0 points)

`public static int length(char[] arr)` (5 points)

`public static char charAt(char[] arr, int index)` (5 points)

`public static char[] concat(char[] a, char[] b)` (5 points)

`public static char[] replace(char[] arr, char c, char replace)` (5 points)

`public static char[] substring(char[] arr, int index)` (15 points)

`public static char[] substring(char[] arr, int begin, int end)` (15 points)

`public static int compareTo(char[] a, char[] b)` (15 points)

`public static char[] toLowerCase(char[] arr)` (10 points)

`public static int compareToIgnoreCase(char[] a, char[] b)` (10 points)

`public static int indexOf(char[] arr, char c)` (5 points)

`public static int indexOf(char[] arr, char[] sub)` (10 points)

We recommend to implement the functions in the order in which they are listed here (and in the given class skeleton).

Lexicographic order

The `compareTo` function checks if one char array is greater than, equal to, or less than another char array, in terms of their *lexicographic order*. Any two strings, as well as any two char arrays, can be compared lexicographically, which is essentially that same as dictionary order. For example:

`[animal]` < `[ant]` < `[dog]` < `[duck]` < `[duckling]` < `[eel]` < `[elephant]`

Note that if two char arrays contain the same characters in the same positions, then the shorter array is lexicographically less than the longer array.

This is the same as dictionary order, with one exception: Uppercase letters are less than lowercase letters. For example, `[duck]` > `[Duck]`, and `[ant]` > `[Zebra]`. This is because all the uppercase letters have lower ASCII values than all the lowercase letters.

Submission

Zip the single `CharArray.java` class into a file named `hw4.zip`, and submit it by following the submission instructions given in Moodle.

Get feedback: To get feedback about your programs before submitting them, use GETFEED, as many times as you want.

Deadline: Submit Homework 4 no later than Sunday, November 29, 2020, 23:55. You are welcome to submit earlier.