Hello, I know we are not supposed to use any standard library functions, but I was wondering if we were still allowed to use #include <string> for this assignment?

<string> is part of the C++ Standard Library, so it's off-limits here.

(Think about it this way: What you're building is basically std::string. If you were to use std::string, there would be no point in doing the assignment.)

Yes, you will need dynamically-allocated arrays.

Hello, If the substring being passed is an empty string, is the contains function supposed to return true or false?

True. All strings contain the empty string.

Hi, are we supposed to edit the .hpp file or leave as is?

Leave as-is.

(Remember that a lot of how this is being graded is via unit tests. Changes you make to the design mean that the unit tests won't compile.)

When the documentation says that the given char array is assumed to be null terminated, does that mean that the last pointer of the array is the null pointer?

Not exactly. It means that there's an extra character on the end of the array, the null character (whose character code is

How would this function:

char& at(unsigned int index)

be used to modify a string, if the only argument passed is an index? What would we be changing the character at the index to?

It returns a reference to a character. Using that reference, the caller could modify the character. (So it's not **at()** that modifies it.)

Check out the unit test named **canAssignIndividualCharactersInNonConstString** in **StringTests.cpp** for an example of how that works.

This is an assignment of a pointer to a constant value (i.e., one that promises not to allow you to change the value stored where it points) *into* a pointer to a non-constant one (i.e., one that doesn't make that same promise). That's illegal for good reason: It would allow you to violate the constness of what **chars** is pointing to.

Since toChars() wants us to return a C-style array, does that mean it must be null terminated?

Yes. A C-style string is always null-terminated.

Are the Strings supposed to have '\0' at the end of them, similar to the C-style arrays, or are they supposed to end at the char before the '\0'

You'll need to decide how best to implement this.

That said, one thing worth noting is that your **toChars** member function needs to be able to return a C-syle string without dynamically allocating one.

I saw the post that finding empty string in another string would be -1. What if finding empty string in empty string? Would it be still return -1?

Yes.

Empty means empty: no characters.

A string containing spaces is not empty.

If startIndex == endIndex, would it return an empty string?

Yes.

In substring method, although it is marked as "unsigned int" for both startIndex and endIndex, do we have to consider user putting negative integer?
In this case, do we have to throw OutOfBoundsException, or we can ignore this case?
I'm concerned because if we put negative value to unsigned int variable, it changes to garbage value (undefined behavior).

Also, if the startIndex is greater than endIndex, and the startIndex is OutOfBounds but the endIndex is not, we throw Exception (and not return empty string), right?

There is no way to pass a negative value to this member function, because the type is unsigned. So there is nothing to consider here.

It's certainly possible that someone might pass -1 and the compiler might implicitly convert this to something unsigned, but there would be no way for your member function to know that happened. So there's nothing that can reasonably be done about it.

On your other question, yes.