CISP 430 Data Structures
Project 2: Classes

As we have discovered, there are some problems involved in producing a user- defined type that resembles a built-in type using a **`C`** module. With **`C++`** we may design and implement a user defined type that, if properly done, is indistinguishable from a built-in type. For this project you will write user-defined type **`STRING`** using a **`C++`** class. You will be required to have the following methods and friend functions:

Member Functions:

1. **`STRING()`**. This constructor will initialize an empty **`STRING`**.
2. **`STRING(char*)`**. This constructor will take a **`char*`** and use it to initialize the **`STRING`**.
3. **`STRING(char)`**. This constructor will take a **`char`** and use it to initialize the **`STRING`**.
4. **`STRING(STRING)`**. This constructor will take a **`STRING`** and use it to initialize the **`STRING`**. This is called the **copy constructor.**
5. **`~STRING()`**. The destructor.
6. **`length()`**. This will return the number of characters in the **`STRING`**.
7. Assignment (**`=`**) operator. This operator assigns one **`STRING`** to another.
8. **`position(char)`**. This will return the position of the first occurrence of **`char`** in the **`STRING`** as an int. Returns **`-1`** if the **`char`** is not in the **`STRING`**.
9. Immediate concatenation (**`+=`**) operator. This operator will be overloaded to work with a right hand value of either type **`STRING`**, type **`char*`** or type **`char`**.
10. Index (**`[ ]`**) operator. This operator returns one character through indexing. An error is handled if the index is out of range. This is to be overloaded with a **`const`** and non-**`const`** version.[A]
11. **`upcase(unsigned first, unsigned last)`**. This function will change all alphabetic characters to upper case.[**B**]
12. **`downcase(unsigned first, unsigned last)`**. This function will change all alphabetic characters to lower case.
13. **`togglecase(unsigned first, unsigned last)`**. This function will change the case of all alphabetic characters.

A Bounds checks must be done on these to make sure this index is in range. We will discuss what to do if there is an out-of-bounds error.

B Functions 11,12, & 13 will work on the character at index first through, but not including, the character at index last. Bounds checks must be done on these to make sure they are in range. We will discuss what to do if there is an out-of-bounds error.

Extra credit 14-16 (0/1) – they either work perfectly and you get full credit or they don't work perfectly and you don't get any credit. You MUST not use any built-in functions

14. **`operator char*()`**. This will cast our **`STRING`** to a **`char*`**.
15. **`operator int()`**. This will cast our **`STRING`** to an int if possible. It will return 0 if unsuccessful.
16. **`operator float()`**. This will cast our **`STRING`** to a float if possible. It will return 0 if unsuccessful.

For operators **`int()`** and **`float()`**, follow the rules for functions **`atoi`** and **`atof`**.

Friend Functions:

17. Output stream (**<<**) operator. This operator will return an **`ostream`**.
18. Input stream (**>>**) operator. This operator will return an **`istream`**.
19. Comparison (**==**) operator. **(see note below for all comparison operators)**
20. Comparison (**`!=`**) operator.
21. Comparison (**>**) operator.
22. Comparison (**<**) operator.
23. Comparison (**<=**) operator.
24. Comparison (**>=**) operator.
25. Concatenation (**+**) operator.

Operators 19-25 will be overloaded to work with these combinations:

- `STRING op STRING`
- `STRING op` **`char`**
- **`char`** `op STRING`
- **`char*`** `op STRING`
- `STRING op` **`char*`**

Comparison Operators:

Two `STRING`s will be said to be equal if they contain exactly the same characters, for example, **`"abc" == "abc"`** but **`"ABC" != "abc"`**. The operators, "**>**", "**<**", "**>=**", and "**<=**" will compare the `STRING`s alphabetically, for example, "**`a`**" **<** "**`b`**", "**`A`**" **<** "**`B`**", "**`A`**" **<** "**`a`**", "**`a`**" **<** "**`B`**", and "**`ab`**" **<** "**`abc`**". Notice that the numerical (ASCII) value of a character decides its position when comparing upper to lower case of the same letter, e.g., "**`A`**" **<** "**`a`**", but it does not when comparing differing letters, e.g., "**`a`**" < "**`B`**". All non-alphabetic characters will be compared by their **`ASCII`** values.

YOU ARE TO IMPLEMENT A MINIMUM NUMBER (2) OF THE COMPARISON OPERATORS AND USE THESE TO IMPLEMENT THE OTHERS. IMPLEMENT OPERATOR== AND OPERATOR>. USE THESE TO IMPLEMENT ALL OF THE OTHER COMPARISON OPERATORS.

`STRING`s must be able to handle any length without using more space than is needed. This will require that you dynamically allocate and deallocate memory. You must be very careful to make sure that any memory that is being used has been allocated and any memory that will no longer be used is deallocated.

YOU MUST NOT DEPEND ON THE NULL-TERMINATOR; IT MUST BE TREATED THE SAME AS ANY OTHER CHARACTER IN YOUR STRING. THERE ARE 256 CHARS

THERE ARE NO SPECIAL CHARACTERS

However, you cannot control how built-in insertion and extraction operators treat the null-terminator – so don't worry about it
For all functions you will need to decide how the parameters should be passed and what should be returned.
I have not included either `const` or `&` in my specifications; you must decide when and where to do so. Member functions such as **`length`** and **`position`,** will be named as in the specifications. Make any function **`const`** that can be made **`const`**.

Each function will have appropriate **`PRE`** and **`POST`** conditions written for it. You may group similar function under one **`PRE`** condition if appropriate. You must write class invariants. All parameters will be labeled, **`/*in*/`**, **`/*out*/`**, or **`/*inout*/`**.

Again: these are a major part of this assignment

Implement the class and prepare a test driver. **Please reuse as much of the code from project 1 as you can for the test driver.**

Note: the final project will consist of **three files**, the class specification (**`.h`**) file, the class implementation file(**`.cpp`**), and the test driver file.

MyString.h
MyString.cpp
TestMain.cpp

You **MUST NOT** use any **`C/C++`** built-in string manipulation functions in your implementation! However, you can and should use your own functions in the implementation of others**.**