CS205 – Lab Assignment 5

Note:

For this lab assignment, you **ONLY** upload `UTF8string.cpp` and `UTF8string.hpp`.

You **MUST NOT** modify `utf8.h` and `utf8.c` because they are external libraries.

You **MUST** use `std::string` as a member variable to store the string, and you are recommended to use its member functions to make your code clearer.

You **MUST NOT** use `std::u16string` or `std::u32string` because you are required to use the `utf8.c` library to deal with Unicode.

Your code **MUST** pass the given test program and have the expected output.

Please write necessary comments for your code.

Part 1

This lab will use the UTF8 functions that you are now familiar with and will make you combine C and C++.

You are asked to create a class called UTF8string; the difference between UTF8string and a regular C++ string is that UTF8string knows "characters" when a string only knows bytes.

The following is provided to simplify your work:

- Test program (testUTF8string.cpp)

You must also use utf8.c and utf8.h. However, you should note that some modifications are required for the C++ compiler to know that the code needs to be compiled in C (C and C++ are incompatible in various ways).

The utf8.c and utf8.h provided in sakai have already been modified.

```
#ifdef __cplusplus
extern "C" {
#endif
extern   int   utf8_charlen(unsigned char *p);
extern   int   utf8_bytes_to_charpos(unsigned char *s, int pos);
extern ...
#ifdef __cplusplus
}
#endif
```

Because rules for finding the right function (the technical name is "resolving") are different in C and C++, this is required to tell the linker that these are C, not C++, functions and that C rules should apply.

You mustn't derive the class from the string class (which wasn't designed as a base class); however, you should use a string attribute to store the string.

You are asked to write the four following methods:

- length(), that returns the length IN CHARACTERS of the UTF8string
- bytes(), that returns the number of bytes used for storing the UTF8string
- find(string substr), that returns the CHARACTER POSITION where substr starts.

  For instance, in "Mais où sont les neiges d'antan", find() should find that "sont" starts at character 8, even if 'ù' is stored on two bytes.
- replace(UTF8string to_remove, UTF8string replacement), that replaces to_remove with replacement.

You'll have to mix C (char *) strings with the C++ std::string type. It's fairly easy to switch between both; there is a constructor that constructs a string from a char * C string passed as parameter; and the method c_str() applied to a C++ std::string returns a pointer to a '\0' terminated sequence of C chars.

Part 2

We'll extend the UTF8string class by adding overloaded operators.

You are asked to redefine:

<< i.e. support std::cout << ustr << std::endl;

+ that gives regular concatenation (if two objects are called u1 and u2, u1 + u2 changes neither u1 nor u2)

+= to append another string (u1 += u2 changes u1, not u2)

* for repeating a string n times (if u is "àéèç", u * 2 or 2 * u should return "àéèçàéèç" without changing u)

! for reversing a string (without modifying original string), which means reversing the characters (not the bytes!), for instance if u is "étudiant" (student in French), !u should be "tnaiduté".