**CS205 C/ C++ Programming Assignment5**

**Name**: 何宜芮(He Yirui)

**SID**: 11811031

**Part 1 – Analysis**

This Assignment first requires us compile C and C++ together, and the assignment paper have already told us how to rewrite the header file.

**Part1:**

Then we need to write length(), bytes(), find(string substr), replace(UTF8string to\_remove, UTF8string replacement) four functions.

1. length(): We can use the function “utf8\_to\_codepoint” in utf8.h to count the length of the UTF8string.
2. bytes(): We can observe the method “.length()” of string class can return byte of a string. So we can use it directly.
3. find(string substr): First, we can use “.find()” in string class to find whether the substring is in the string or not. If we did not find, that means there is no such substring in the string. If we find, the position “.find()” in string class returns means how many bytes before the substring, so we use “utf8\_to\_codepoint” again to find the exact position.(However, this method can only find one position, so the return position is where we first find the substring)
4. replace(UTF8string to\_remove, UTF8string replacement): We use find method we designed before to find the substring, replace them until we can’t find such substring.

**Part2:**

This part is asking us to redefine some operators.

1. << : Since we need to output other things at the same time, so we use friend function to redefine it. Since it will not modify the object of UTF8string, we can define it as a const function.
2. + : Since it will not modify the object of UTF8string, we can define it as a const function. We need to add two objects, and return a new one. So we need to return a copy of the new object.
3. +=: Since we need to modified one of the object, so we write it as a class function.
4. \*: Without changing the origin string, we need to repeat it many times. Since the input n can be any integer, we use for loop to deal with it. We need to redefine two kinds of function since 2\*u and u\*2 are both valid.
5. !: It is an operator acting on only one object without modifying it, so we write the redefine function as a class function. And in the new string, we need to append one character after a character before it.

**Part 2 – Code**

**.hpp file**

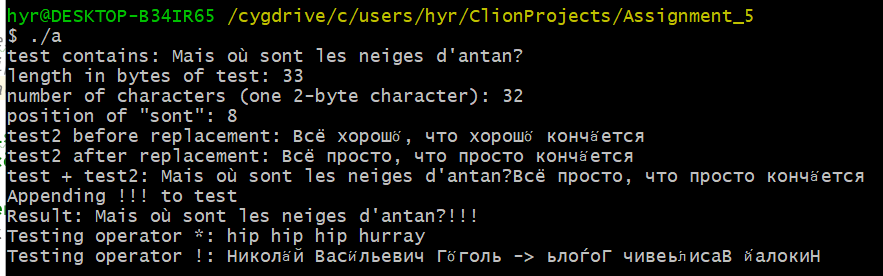
#ifndef **ASSIGNMENT\_5\_UTF8STRING\_HPP**#define **ASSIGNMENT\_5\_UTF8STRING\_HPP**#include **<string>  
  
  
class** UTF8string {  
**private**:  
 std::string str;  
  
**public**:  
 UTF8string(std::string s);  
  
 UTF8string(**char** s[]);  
  
 UTF8string();  
  
 size\_t length() **const**;  
  
 size\_t bytes() **const**;  
  
 **long** find(std::string substr) **const**;  
  
 **void** replace(UTF8string to\_remove, UTF8string replacement);  
  
 std::string getStr() **const**;  
  
 **friend** std::ostream &**operator**<<(std::ostream &os, **const** UTF8string &s);  
  
 **friend** UTF8string **operator**+(**const** UTF8string &s1, **const** UTF8string &s2);  
  
 **friend void operator**+=(UTF8string &s1, **const** UTF8string &s2);  
  
 **friend** UTF8string **operator**\*(**const** UTF8string &s, **int** time);  
  
 **friend** UTF8string **operator**\*(**int** time, **const** UTF8string &s);  
  
 UTF8string **operator**!() **const**;  
  
};  
  
  
#endif *//ASSIGNMENT\_5\_UTF8STRING\_HPP*

**.cpp file**

#include **"UTF8string.hpp"**#include **<cstring>**#include **<string>**#include **"utf8.h"**#include **<iostream>  
  
using namespace** std;  
  
string UTF8string::getStr() **const** {  
 **return** str;  
}  
  
UTF8string::UTF8string(string s) {  
 str = s;  
}  
  
UTF8string::UTF8string() {  
 str = **""**;  
}  
  
UTF8string::UTF8string(**char** s[]) {  
 str = string(s);  
}  
  
size\_t UTF8string::length() **const** {  
 **char const** \*input = str.c\_str();  
 **unsigned long** codepoint;  
 **int** bytes\_in\_char;  
 size\_t length = 0;  
  
 **unsigned char** \*p;  
 p = (**unsigned char** \*) (&input[0]);  
 **while** (\*p) {  
 codepoint = utf8\_to\_codepoint(p, &bytes\_in\_char);  
 **if** (codepoint) {  
 length++;  
 **\_utf8\_incr**(p);  
 } **else** {  
 p++;  
 }  
 }  
 **return** length;  
}  
  
size\_t UTF8string::bytes() **const** {  
 **return** str.length();  
  
}  
  
**long** UTF8string::find(string substr) **const** {  
 **long** find = str.find(substr);  
 **if** (find == string::npos)  
 **return** find;*//not find* **char const** \*input = str.substr(0,find).c\_str();  
 **unsigned long** codepoint;  
 **int** bytes\_in\_char;  
 **unsigned long** result = 0;  
  
 **unsigned char** \*p;  
 p = (**unsigned char** \*) (&input[0]);  
 **while** (\*p) {  
 codepoint = utf8\_to\_codepoint(p, &bytes\_in\_char);  
 **if** (codepoint) {  
 result++;  
 }  
 **\_utf8\_incr**(p);  
 }  
 **return** result;  
  
}  
  
**void** UTF8string::replace(UTF8string to\_remove, UTF8string replacement) {  
 **while** (str.find(to\_remove.getStr()) != string::npos)  
 str = str.replace(str.find(to\_remove.getStr()), to\_remove.bytes(), replacement.getStr());  
 **return**;  
}  
  
ostream &**operator**<<(ostream &os, **const** UTF8string &s) {  
 **return** os << s.str;  
}  
  
UTF8string **operator**+(**const** UTF8string &s1, **const** UTF8string &s2) {  
 UTF8string u(s1.str + s2.str);  
 **return** u;  
}  
  
**void operator**+=(UTF8string &s1, **const** UTF8string &s2) {  
 s1.str = s1.str + s2.str;  
}  
  
  
UTF8string **operator**\*(**const** UTF8string &s, **int** time) {*//if time<0,it will return “”.* UTF8string u;  
 **for** (**int** i = 1; i <= time; i++) {  
 u.str = u.str + s.str;  
 }  
 **return** u;  
}  
  
UTF8string **operator**\*(**int** time, **const** UTF8string &s) {  
 **return** s \* time;  
}  
  
UTF8string UTF8string::**operator**!() **const** {  
 **char const** \*input = str.c\_str();  
 string result = **""**;  
 **int** bytes\_in\_char;  
 **unsigned long** codepoint;  
  
 **unsigned char** \*p;  
 p = (**unsigned char** \*) (&input[0]);  
 **while** (\*p) {  
 codepoint = utf8\_to\_codepoint(p, &bytes\_in\_char);  
 **if** (codepoint) {  
 string block = **""**;  
 **for** (**int** i = 0; i < bytes\_in\_char; ++i) {  
 block += p[i];  
 }  
 result = block + result;  
 p += bytes\_in\_char;  
 } **else** {  
 p++;  
 }  
 }  
 UTF8string resultUTF8(string);  
 **return** result;  
}

**Part 3 - Result & Verification**

**Test program on sakai:**

****

**It is the same as the given result.**

**Part 4 - Difficulties & Solutions**

1. First, I use an array of char to inverse a string. However, blanks are garbled. Finally I gave up using C-style string and used the string in C++. Although it is not quick, it works.
2. First I do not know how to redefine “<<” correctly, so I read the textbook and get the right answer.