CS205 C/ C++ Programming - Lab Assignment 5

Name: 邱煜（Qiu Yu）

SID: 11611127

Part1 - Analysis

This assignment is to create a class called `UTF8string` which knows "characters". I use the functions in `uf8.c` to implements these required functions.

Part2 - Code

UTF8string.cpp

```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <vector>
#include <algorithm>
#include "UTF8string.hpp"
#include "utf8.h"

using namespace std;

// empty constructor
UTF8string::UTF8string(){
    str = "";
    size = 0;
    byte = 0;
}

// only string constructor
UTF8string::UTF8string(string rawStr) {
    str = rawStr;
    byte = rawStr.length();

    int lenPtr = 0;
    int readLen = 0;
    int codePt;
    unsigned char *pt = (unsigned char *)str.c_str();
    size = 0;
    while (readLen < byte)
```

```
    {
        lenPtr = 0;
        codePt = utf8_to_codepoint(pt, &lenPtr);
        pt += lenPtr;
        readLen += lenPtr;
        size++;
        eachChar.push_back(codePt);
        byteNum.push_back(lenPtr);
    }
}

UTF8string::UTF8string(const char rawStr[]){
    str = rawStr;
    byte = str.length();

    int lenPtr = 0;
    int readLen = 0;
    int codePt;
    unsigned char *pt = (unsigned char *)str.c_str();
    size = 0;
    while (readLen < byte) {
        lenPtr = 0;
        codePt = utf8_to_codepoint(pt, &lenPtr);
        pt += lenPtr;
        readLen += lenPtr;
        size++;
        eachChar.push_back(codePt);
        byteNum.push_back(lenPtr);
    }
}

// partly constructor
UTF8string::UTF8string(string rawStr, int rawSize, int
rawByte){
    str = rawStr;
    size = rawSize;
    byte = rawByte;

    int lenPtr = 0;
    int readLen = 0;
    int codePt;
    unsigned char *pt = (unsigned char *)str.c_str();
    while (readLen < byte) {
        lenPtr = 0;
        utf8_to_codepoint(pt, &lenPtr);
        pt += lenPtr;
        readLen += lenPtr;
```

```
        eachChar.push_back(codePt);
        byteNum.push_back(lenPtr);
    }
}

//full constructor
UTF8string::UTF8string(string rawStr, int rawSize, int
rawByte, vector<int> rawByteNum, vector<int> rawEachChar){
    str = rawStr;
    size = rawSize;
    byte = rawByte;
    byteNum = rawByteNum;
    eachChar = rawEachChar;
}

// overload <<
ostream &operator<<(ostream & os, const UTF8string &
utf8string){
    os << utf8string.str;
    return os;
}

// overload +
UTF8string UTF8string::operator+(const UTF8string &a) {
    UTF8string result(this->str+a.str, this->size+a.size,
this->byte+a.byte);
    result.byteNum.reserve(result.size+a.size+1);
    result.byteNum.insert(result.byteNum.end(),
a.byteNum.begin(), a.byteNum.end());
    result.eachChar.reserve(result.size+a.size+1);
    result.eachChar.insert(result.eachChar.end(),
a.eachChar.begin(), a.eachChar.end());
    return result;
}

// overload +=
void UTF8string::operator+=(const UTF8string &a){
    *this = *this + a;
}

// overload *
UTF8string UTF8string::operator*(const int num) {
    UTF8string result;
    for (int i = 0; i < num; i++) {
        result += *this;
    }
    return result;
```

```
}
UTF8string operator*(int num, const UTF8string &str) {
    UTF8string result;
    for (int i = 0; i < num; i++) {
        result += str;
    }
    return result;
}

// overload !
UTF8string UTF8string::operator!(){
    UTF8string result("", this->size, this->byte, this->byteNum, this->eachChar);
    reverse(result.eachChar.begin(), result.eachChar.end());
    reverse(result.byteNum.begin(), result.byteNum.end());
    unsigned char *p = (unsigned char *)malloc(sizeof(unsigned char) * 5);
    for (int i = 0; i < size; i++) {
       result.str += (char *)codepoint_to_utf8(result.eachChar[i], p);
    }
    return result;
}

// return character length
int UTF8string::length() { return size; }

// return byte length
int UTF8string::bytes() { return byte; }

// return the character position where substr starts. -1 if not found
int UTF8string::find(string substr){
    int charIdx = 0;
    int nowByte = 0;
    int byteIdx = str.find(substr);
    if(byteIdx == -1){ return byteIdx; }
    for(int i = 0; i < size; i++){
        if(nowByte < byteIdx){
            nowByte += byteNum[i];
            charIdx++;
        }else{ break; }
    }
    return charIdx;
}

// replace the to_remove with replacement
```

```
int UTF8string::replace(UTF8string to_remove, UTF8string
replacement){
    int byteIdx = str.find(to_remove.str);
    if(byteIdx == -1){ return byteIdx; }
    str = str.replace(byteIdx, to_remove.str.length(),
replacement.str);

    size -= to_remove.length();
    size += replacement.length();
    byte -= to_remove.bytes();
    byte += replacement.bytes();

    int lenPtr = 0;
    int readLen = 0;
    int codePt;
    unsigned char *pt = (unsigned char *)str.c_str();
    while (readLen < byte) {
        lenPtr = 0;
        codePt = utf8_to_codepoint(pt, &lenPtr);
        pt += lenPtr;
        readLen += lenPtr;
        eachChar.push_back(codePt);
        byteNum.push_back(lenPtr);
    }
    return 1;
}
```

H3 UTF8string.hpp

```
#ifndef UTF8string_hpp
#define UTF8string_hpp

#include <vector>

class UTF8string {
   public:
    std::string str;
    int size;
    int byte;
    std::vector<int> byteNum;
    std::vector<int> eachChar;

    // constructor and destructor
    UTF8string();
    UTF8string(std::string);
```

```
17      UTF8string(const char[]);
18      UTF8string(std::string, int, int);
19      UTF8string(std::string, int, int, std::vector<int>,
    std::vector<int>);
20      ~UTF8string(){};
21
22      // overload operator
23      friend std::ostream &operator<<(std::ostream &, const
    UTF8string &);
24      UTF8string operator+(const UTF8string &);
25      void operator+=(const UTF8string &);
26      UTF8string operator*(const int);
27      friend UTF8string operator*(int, const UTF8string &);
28      UTF8string operator!();
29
30      // member functions
31      int length();
32      int bytes();
33      int find(std::string);
34      int replace(UTF8string, UTF8string);
35  };
36
37  #endif
```

H2 Part 3 - Result & Verification

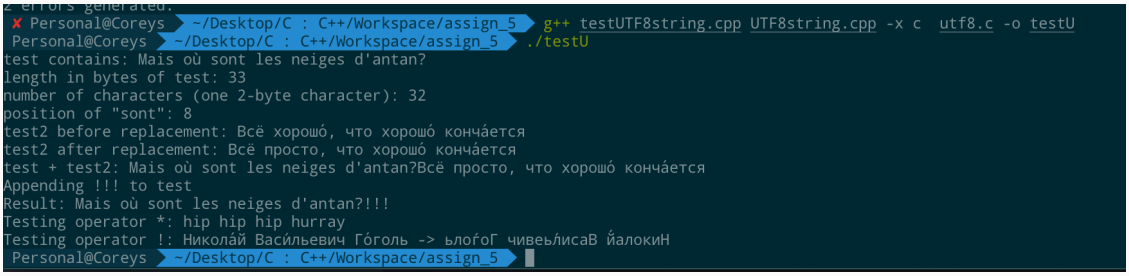
H3 Test case

Test case #1:

```
1   Input:
2       testUTF8string.cpp
3
4   Output:
5       test contains: Mais où sont les neiges d'antan?
6       length in bytes of test: 33
7       number of characters (one 2-byte character): 32
8       position of "sont": 8
9       test2 before replacement: Всё хорошó, что хорошó
    кончáется
10      test2 after replacement: Всё просто, что хорошó
    кончáется
11      test + test2: Mais où sont les neiges d'antan?Всё
    просто, что хорошó кончáется
```

```
    Appending !!! to test
    Result: Mais où sont les neiges d'antan?!!!
    Testing operator *: hip hip hip hurray
    Testing operator !: Никола́й Васи́льевич Го́голь -> ьлоґоГ чивеьли́саВ йа́локиН

Verification:
test contains: Mais où sont les neiges d'antan?
length in bytes of test: 33
number of characters (one 2-byte character): 32
position of "sont": 8
test2 before replacement: Всё хорошо́, что хорошо́ конча́ется
test2 after replacement: Всё просто, что просто конча́ется
test + test2: Mais où sont les neiges d'antan?Всё просто, что просто конча́ется
Appending !!! to test
Result: Mais où sont les neiges d'antan?!!!
Testing operator *: hip hip hip hurray
Testing operator !: Никола́й Васи́льевич Го́голь -> ьлоґоГ чивеьли́саВ йа́локиН
```



H2 Part 4 - Difficulties & Solutions

1. We need to store the byte information of the UTF8string. I use two `vector` to implements that. One stores the byte length of each character of the string, one stores each the code point of each character.