HOMEWORK 5

- String.h의 String 클래스를 구현하는 엘리먼트(operators)을 작성하여 프로그램을 완성하시오.

```cpp
#include <iostream>
using std::ostream;
using std::istream;

class String {

  friend ostream &operator<<( ostream & out, const String & );
  friend istream &operator>>( istream & in, String & );

public:
  String( const char * = "" ); // conversion/default constructor
  String( const String & ); // copy constructor
  ~String(); // destructor

  const String &operator=( const String & ); // assignment operator
  const String &operator+=( const String & ); // concatenation operator

  bool operator!() const; // is String empty?
  bool operator==( const String & ) const; // test s1 == s2
  bool operator<( const String & ) const; // test s1 < s2

  // test s1 != s2
  bool operator!=( const String &right ) const ;
  // test s1 > s2
  bool operator>( const String &right ) const ;
  // test s1 <= s2
  bool operator<=( const String &right ) const ;
  // test s1 >= s2
  bool operator>=( const String &right ) const ;

  char &operator[]( int ); // subscript operator (modifiable lvalue)
  char operator[]( int ) const; // subscript operator (rvalue)
```
```cpp
String operator()( int, int = 0 ) const; // return a substring
int getLength() const; // return string length

private:
    int length; // string length (not counting null terminator)
    char *sPtr; // pointer to start of pointer-based string

    void setString( const char * ); // utility function
}; // end class String

#include <iostream>
using std::cout;
using std::endl;
using std::boolalpha;

#include "String.h"

int main() {

    String s1( "happy" );
    String s2( " birthday" );
    String s3;

    // test overloaded equality and relational operators
    cout << "s1 is W"" << s1 << "W"; s2 is W"" << s2
        << "W""; s3 is W"" << s3 << 'W''
        << boolalpha << "WhWhThe results of comparing s2 and s1:""<< "Wns2 == s1 yields " << ( s2 == s1 )
        << "Wns2 != s1 yields " << ( s2 != s1 )
        << "Wns2 > s1 yields " << ( s2 > s1 )
        << "Wns2 < s1 yields " << ( s2 < s1 )
        << "Wns2 >= s1 yields " << ( s2 >= s1 )
        << "Wns2 <= s1 yields " << ( s2 <= s1 );
```
// test overloaded String empty (!) operator
cout << "### Testing !s3: " << endl;

if( !s3 ) {
    cout << "s3 is empty; assigning s1 to s3:" << endl;
s3 = s1; // test overloaded assignment
    cout << "s3 is " << s3 << " ";
} // end if

// test overloaded String concatenation operator
cout << "### s1 += s2 yields s1 = ";
s1 += s2; // test overloaded concatenation
    cout << s1;

// test conversion constructor
cout << "### s1 += " to you " yields " << endl;
s1 += " to you"; // test conversion constructor
    cout << "s1 = " << s1 << " ";

// test overloaded function call operator () for substring
cout << "### The substring of s1 starting at 0 location 0 for 14 characters, s1(0, 14), is: 
     " << s1( 0, 14 ) << " 
     ";

// test substring "to-end-of-String" option
cout << "### The substring of s1 starting at 0 location 15, s1(15), is: 
      " << s1( 15 ) << " 
     ";

// test copy constructor
String *s4Ptr = new String( s1 );
    cout << "### *s4Ptr = " << *s4Ptr << " 
     ";

// test assignment (=) operator with self-assignment
cout << "### assigning *s4Ptr to *s4Ptr " << endl;
*s4Ptr = *s4Ptr; // test overloaded assignment
cout << "*s4Ptr = " << *s4Ptr << endl;

// test destructor
delete s4Ptr;

// test using subscript operator to create a modifiable lvalue
s1[ 0 ] = 'H';
s1[ 6 ] = 'B';
cout << "ns1 after s1[0] = 'H' and s1[6] = 'B' is:" << "s1 = " << s1 << endl;

// test subscript out of range
cout << "Attempt to assign 'd' to s1[30] yields:" << endl;
s1[ 30 ] = 'd'; // ERROR: subscript out of range
return 0;
}
} // end main