

## Übung 13

infT

Es ist eine eigene Klasse `string` zu entwerfen, mit der `string`-Objekte erzeugt und manipuliert werden können sollen. Die Klasse bediene sich des Operator Overloading.

Gegeben ist das Header-File:

```

/* String1.h */
#ifndef STRING1_H
#define STRING1_H

#include <iostream>
using namespace std;

class String {
    int length;                // string length
    char *sPtr;               // pointer to start of string

    void setString(const char *); // utility function

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

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

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

    char& operator[](int);        // subscript operator
    const char& operator[](int) const; // subscript operator
    String operator()(int, int);  // Return a substring
    int getLength() const;       // Return string length

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

#endif

```

Intern werde ein `String` durch ein dynamisch alloziertes `char`-Array repräsentiert, auf das der Pointer `sPtr` verweise. Die Länge des strings (ohne Delimiter '`\0`') werde in `length` gehalten.

Implementieren Sie das File `String1.cpp`, so dass das gegebene `String1Test.cpp`-File ausgeführt werden kann.

Hinweise:

- Der Operator ! teste, ob ein String leer ist oder nicht. D.h.  

```
String s; // instantiate an empty string
cout << !s ? "true" : "false";
```

liefert true an cout.
- Falls Subscript zu klein oder zu gross: Fehlermeldung und Programmabbruch.
- Self-Assignment verhindern.
- Realisierung von inline-Functions (im Header-File), wo sinnvoll.
- Memory Leaking verhindern.
- Die Ausgabe an cout laute:

```
Conversion constructor: happy
Conversion constructor: birthday
Conversion constructor:
s1 is "happy"; s2 is " birthday"; s3 is ""
```

The results of comparing s2 and s1:

```
s2 == s1 yields false
s2 != s1 yields true
s2 > s1 yields false
s2 < s1 yields true
s2 >= s1 yields false
s2 <= s1 yields true
```

Testing !s3:

```
s3 is empty; assigning s1 to s3;
operator= called
s3 is "happy"
```

```
s1 += s2 yields s1 = happy birthday
```

```
s1 += " to you" yields
Conversion constructor: to you
Destructor: to you
s1 = happy birthday to you
```

```
Conversion constructor: happy birthday
Copy constructor: happy birthday
Destructor: happy birthday
The substring of s1 starting at
location 0 for 14 characters, s1(0, 14), is:
happy birthday
```

```
Destructor: happy birthday
Conversion constructor: to you
Copy constructor: to you
Destructor: to you
The substring of s1 starting at
location 15, s1(15, 0), is: to you
```

```
Destructor: to you
Copy constructor: happy birthday to you
```

```
*s4Ptr = happy birthday to you
```

```
assigning *s4Ptr to *s4Ptr
operator= called
Attempted assignment of a String to itself
*s4Ptr = happy birthday to you
Destructor: happy birthday to you
```

```
s1 after s1[0] = 'H' and s1[6] = 'B' is: Happy Birthday to you
```

```
Attempt to assign 'd' to s1[30] yields:
Error: Subscript 30 out of range
```