

CS 11 Data Structures and Algorithms

Assignment 5: Dynamic Memory in Classes

[Return to Course Homepage](#)

Assignment 5.1

```

/*
Name: Sam Student
Date: Sept. 29, 2015
Assignment Number: 6
Instructor: Dave Harden
File: mystring.h

The MyString class is designed to make working with strings easier and less
error-prone than working with traditional null-terminated C-strings. The client
can declare and use MyStrings freely without concern for memory management
issues or the size of the MyString. Operations for input/output, construction,
indexing, comparison, and concatenation of MyStrings are provided. Assignment
and copying of MyString objects is allowed.

MyString(const char* inString);
post: a MyString object is created and initialized to "inString".

MyString();
post: a MyString object is created and initialized to the empty string.

MyString(const MyString& copyMe);
post: a MyString object is created and initialized to "copyMe".

friend ostream& operator<<(ostream& out, const MyString& printMe);
pre: "out" is ready for writing.
post: The contents of "printMe" have been inserted into "out".

char operator[] (int index) const;
pre: 0 <= index < length()
post: The character at position "index" (counting from 0) has been returned.

char& operator[](int index);
pre: 0 <= index < length()
post: The character at position "index" (counting from 0) has been returned.

friend bool operator<(const MyString& left, const MyString& right);
post: true is returned if left < right; false otherwise.

friend bool operator>(const MyString& left, const MyString& right);
post: true is returned if left > right; false otherwise.

friend bool operator<=(const MyString& left, const MyString& right);
post: true is returned if left <= right; false otherwise.

friend bool operator>=(const MyString& left, const MyString& right);
post: true is returned if left >= right; false otherwise.

friend bool operator==(const MyString& left, const MyString& right);
post: true is returned if left == right; false otherwise.

friend bool operator!=(const MyString& left, const MyString& right);
post: true is returned if left != right; false otherwise.

MyString operator=(const MyString& right);
post: A copy of "right" is stored in the calling object.

int length() const;
post: the number of characters in the calling object is returned.

*/

#ifndef MYSTRING_H
#define MYSTRING_H

```

```

#include <iostream>
namespace compsci_mystring{
    class MyString {
    public:
        MyString(const char* inString);
        MyString();
        MyString(const MyString& copyMe);
        ~MyString();
        friend std::ostream& operator<<(std::ostream& out, const MyString& printMe);
        char operator[] (int index) const;
        char& operator[](int index);
        friend bool operator<(const MyString& left, const MyString& right);
        friend bool operator>(const MyString& left, const MyString& right);
        friend bool operator<=(const MyString& left, const MyString& right);
        friend bool operator>=(const MyString& left, const MyString& right);
        friend bool operator==(const MyString& left, const MyString& right);
        friend bool operator!=(const MyString& left, const MyString& right);
        MyString operator=(const MyString& right);
        int length() const;
    private:
        char *str;
    };
}

```

```
#endif
```

```
/*
```

```

Name: Sam Student
Date: Sept. 29, 2015
Assignment Number: 6
Instructor: Dave Harden
File: mystring.cpp

```

```
CLASS INVARIANT:
```

The class has one private data member defined as follows:

```
char *str;
```

str always represents a valid null-terminated c-string

```
*/
```

```

#include "mystring.h"
#include <iostream>
#include <cstring>
#include <cassert>
using namespace std;

```

```

namespace compsci_mystring{
    MyString::MyString(const char* inString)
    {
        str = new char[strlen(inString) + 1];
        strcpy(str, inString);
    }

```

```

    MyString::MyString()
    {
        str = new char[1];
        strcpy(str, "");
    }

```

```

    MyString::MyString(const MyString& copyMe)
    {
        str = new char[strlen(copyMe.str) + 1];
        strcpy(str, copyMe.str);
    }

```

```
}
```

```
MyString::~MyString()  
{  
    delete [] str;  
}
```

```
ostream& operator<<(ostream& out, const MyString& printMe)  
{  
    out << printMe.str;  
    return out;  
}
```

```
char MyString::operator[](int index) const  
{  
    assert (index >= 0 && index < strlen(str));  
    return str[index];  
}
```

```
char& MyString::operator[](int index)  
{  
    assert (index >= 0 && index < strlen(str));  
    return str[index];  
}
```

```
bool operator<(const MyString& left, const MyString& right)  
{  
    return strcmp(left.str, right.str) < 0;  
}
```

```
bool operator>(const MyString& left, const MyString& right)  
{  
    return strcmp(left.str, right.str) > 0;  
}
```

```
bool operator<=(const MyString& left, const MyString& right)  
{  
    return strcmp(left.str, right.str) <= 0;  
}
```

```
bool operator>=(const MyString& left, const MyString& right)
```

```
{
    return strcmp(left.str, right.str) >= 0;
}

bool operator==(const MyString& left, const MyString& right)
{
    return strcmp(left.str, right.str) == 0;
}

bool operator!=(const MyString& left, const MyString& right)
{
    return strcmp(left.str, right.str) != 0;
}

MyString MyString::operator=(const MyString& right)
{
    if (this != &right){
        delete [] str;
        str = new char[strlen(right.str) + 1];
        strcpy(str, right.str);
    }
    return *this;
}

int MyString::length() const
{
    return strlen(str);
}
}
```

© 1999 - 2018 Dave Harden