CS 11 Data Structures and Algorithms

Assignment 6: Dynamic Memory in Classes 2

Return to Course Homepage

Assignment 6.1

```
/*

Name:
Date:
Assignment Number:
Instructor:
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".

friend istream& operator>>(istream& in, MyString& readMe);
pre: "in" is ready for reading.  The sequence of characters read must be fewer
than 128 in number.
post: Leading whitespace in "in" has been skipped and the following sequence of
non-whitespace characters have been extracted from "in" and stored in "readMe".
Reading is terminated by the next whitespace character.

void read(istream& in, char delimeter);
pre: "in" is ready for reading.  The sequence of characters read must be fewer
than 128 in number.
post: The sequence of characters in "in", terminated by "delimiter", have been
extracted and stored in the calling object.

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.

        friend MyString operator+(const MyString& left, const MyString& right);
        post: the concatenation of left and right is returned.

        MyString operator+=(const MyString& right);
        post: the concatenation of left and right is assigned to left and returned.

        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);
                    friend std::istream& operator>>(std::istream& in, MyString& readMe);
                    void read(std::istream& in, char delimeter);
                    static const int MAX_INPUT_SIZE = 127;
                    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);
                    friend MyString operator+(const MyString& left, const MyString& right);
                    MyString operator+=(const MyString& right);
                    int length() const;
                private:
                    char *str;
            };
        }

        #endif


        /*

        Name:
        Date:
        Assignment Number:
        Instructor:
        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);
            }
```

```cpp
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;
}




istream& operator>>(istream& in, MyString& readMe)
{
    delete [] readMe.str;
    char tempStr[MyString::MAX_INPUT_SIZE + 1];
    in >> tempStr;
    readMe.str = new char[strlen(tempStr) + 1];
    strcpy(readMe.str, tempStr);
    return in;
}




void MyString::read(istream& in, char delimiter)
{
    char tempStr[MyString::MAX_INPUT_SIZE + 1];

    in.getline(tempStr, MyString::MAX_INPUT_SIZE + 1, delimiter);
    delete [] str;
    str = new char[strlen(tempStr) + 1];
    strcpy(str, tempStr);
}
```

```cpp
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;
    }




    MyString operator+(const MyString& left, const MyString& right)
    {
        MyString tempStr;
        delete [] tempStr.str;
        tempStr.str = new char[strlen(left.str) + strlen(right.str) + 1];
        strcpy(tempStr.str, left.str);
        strcat(tempStr.str, right.str);
        return tempStr;
    }




    MyString MyString::operator+=(const MyString& right)
    {
        *this = *this + right;
        return *this;
    }




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

© 1999 - 2018 Dave Harden