```
/*
 * mystring.h
 */

#ifndef __MYSTRING_H__
#define __MYSTRING_H__

using namespace std;

#include <iostream>

class MyString {

    public:

        // default constructor
        MyString();

        // constructor
        MyString(const char* p);

        // destructor
        ~MyString();

        // copy constructor
        MyString(const MyString& s);

        // assignment operator
        MyString& operator=(const MyString& s);

        // returns the length of the string
        int length() const { return len; }

        // operator+
        friend MyString operator+(const MyString& s1, const MyString& s2);

        // put-to operator
        friend ostream& operator<<(ostream& os, const MyString& s);

        // get-from operator
        friend istream& operator>>(istream& is, MyString& s);

        // operator[]
        char& operator[](int i);

        // operator[] const
        const char& operator[](int i) const;

    private:

        char* data;

        int len;
};

#endif
```

```
/*
 * mystring.cpp
 */

#include "mystring.h"

// default constructor

MyString::MyString()
{
    data = new char[1];
    data[0] = '\0';

    len = 0;
}

// constructor

MyString::MyString(const char* p)
{
    if (p) {
        len = strlen(p);
        data = new char[len+1];
        strcpy(data, p);
    } else {
        data = new char[1];
        data[0] = '\0';
        len = 0;
    }
}

// destructor

MyString::~MyString()
{
    delete[] data;
}

// copy constructor

MyString::MyString(const MyString& s)
{
    len = s.len;

    data = new char[len+1];
    strcpy(data, s.data);
}

// assignment operator

MyString& MyString::operator=(const MyString& rhs)
{
    if (this == &rhs) {
        return *this;
    }

    // first, deallocate memory that 'this' used to hold
```

```
    delete[] data;

    // now copy from rhs

    len = rhs.len;

    data = new char[len+1];
    strcpy(data, rhs.data);

    return *this;
}

// operator+

MyString operator+(const MyString& s1, const MyString& s2)
{
    MyString temp;

    delete[] temp.data;

    temp.len = s1.len + s2.len;

    temp.data = new char[temp.len+1];
    strcpy(temp.data, s1.data);
    strcat(temp.data, s2.data);

    return temp;
}

// put-to operator

ostream& operator<<(ostream& os, const MyString& s)
{
    os << s.data;
    return os;
}

// get-from operator

istream& operator>>(istream& is, MyString& s)
{
    // this is kinda cheating, but this is just to illustrate how this
    // function can work.

    string temp;
    is >> temp;

    delete[] s.data;

    s.len = strlen(temp.c_str());
    s.data = new char[s.len+1];
    strcpy(s.data, temp.c_str());

    return is;
}

// operator[] - in real life this function should be declared inline
```

```
char& MyString::operator[](int i)
{
    return data[i];
}

// operator[] const - in real life this should be inline

const char& MyString::operator[](int i) const
{
    // illustration of casting away constness
    return ((MyString&)*this)[i];
}
```

```
/*
 * test1.cpp
 */

#include "mystring.h"

int main() {

    MyString s1;

    MyString s2("hello");

    MyString s3(s2);

    s1 = s2;

    cout << s1 << "," << s2 << "," << s3 << endl;

    return 0;
}
```

```
/*
 * test2.cpp
 */

#include "mystring.h"

int main() {

    MyString s1("hello ");

    MyString s2("world!");

    MyString s3;

    s3 = s1 + s2;

    cout << s3 << endl;

    cout << s1 + s2 << endl;

    cout << s1 + "world!" << endl;

    cout << "hello " + s2 << endl;

    // this is an error
    // cout << "hello " + "world!" << endl;

    return 0;
}
```

```
/*
 * test3.cpp
 */

#include "mystring.h"

int main() {

    cout << "Enter a string: ";

    MyString s;

    cin >> s;

    for (int i = 0; i < s.length(); ++i) {

        if (’a’ <= s[i] && s[i] <= ’z’) {
            s[i] = s[i] - (’a’ - ’A’);
        }
    }

    cout << "Here is how to say it louder: " << s << endl;

    return 0;
}
```