面向对象程序设计 Assignment 2

本题需要你按照题目要求实现一个`头文件`，由于需要特殊判断，本题不支持线上自动评测，当作业截至时，将由助教收集所有人的代码进行线下评测，并在群里公布分数。

提交说明：

请将你的代码写在一个命名为`mystring.h`的头文件里，并且这个文件应该只有一个类`MyString`。

请将实现好的`mystring.h`打包成`zip`文件，`zip`文件的命名方式为`学号.zip`，例如你的学号为`12345`，则`zip`文件名称为`12345.zip`，然后使用平台的上传功能上传你的答案。

下面的3个问题你只需要在一个`MyString`里实现，我们会根据你的实现情况给予相应的分数。

测试代码已包含在附件中，你可以先在本地自己评测，评测通过后再上传到平台上。

1

实现一个模仿`std::string`的类`MyString`，底层基于`char *`存储（不允许使用`std::string`，`std::vector`等标准库容器，但可以使用`cstring`中的字符串操作函数），使得以下代码能正常工作且没有内存泄漏或内存重复释放。

```
MyString mystr;
MyString mystr2("a_string");
MyString mystr3(mystr2);

cout << mystr.size() << endl;   // 0
cout << mystr2.size() << endl;  // 8

cout << mystr3 << endl;         // a_string

mystr3[1] = ' ';
cout << mystr3 << endl;         // a string

mystr3[2] = 'b';
cout << mystr3 << endl;         // a btring
```

```
cout << mystr2[1] << endl;       // _

mystr.append("foobar");
cout << mystr.size() << endl;    // 6

mystr2.append("_boxboz");
cout << mystr2 << endl;          // a_string_boxboz

mystr2.append(mystr3);
cout << mystr2 << endl;          // a_string_boxboza btring

cout << mystr3 << endl;          // a btring

cout << (MyString("abc") == MyString("abc")) << endl;  // true
cout << (MyString("abc") <= MyString("abc")) << endl;  // true
cout << (MyString("abd") == MyString("abc")) << endl;  // false
cout << (MyString("abc") != MyString("abc")) << endl;  // false
cout << (MyString("abd") != MyString("abc")) << endl;  // true
cout << (MyString("ab0") < MyString("ab1")) << endl;   // true
cout << (MyString("ab0") > MyString("ab1")) << endl;   // false
cout << (MyString("ab0") >= MyString("ab1")) << endl;  // false
cout << (MyString("15") < MyString("3")) << endl;      // true, because asc

const MyString cstr("a constant string");
cout << cstr.size() << endl;     // 17
cout << cstr[2] << endl;         // c
cout << cstr << endl;            // a constant string
```

2

`std::string` 相比 `char *` 有很多先进之处，其中之一是它不根据 `\0` 判断字符串终结，而是单独存储了一个字段作为长度，这使得它可以避免很多安全问题。你的 `MyString` 能正常存储 `\0` 作为内容吗？如果不能，请修正你的实现。

一个能正常处理 `\0` 的 `MyString` 应当能使得以下代码正常工作：

```
MyString mystr("abc123");
cout << mystr.size() << endl;    // 6

mystr[2] = '\0';
cout << mystr.size() << endl;    // 6

MyString mystr2(mystr);
```

```
cout << mystr2.size() << endl;  // 6

MyString mystr3("foo");
mystr3.append(mystr);
cout << mystr3.size() << endl;  // 9

cout << mystr3 << endl;         // fooab?123
// Note: '\0' may be displayed as different characters
// or even cannot be displayed in different environments.
//
// Here we use '?' to mark its existance.
//
// The important thing here is that "123" should be
// correctly displayed.
```

3

COW (Copy-On-Write) 是一个常见的高级编程技术，可以提升某些情况下的程序运行效率。COW 是写入时复制的意思，即原本需要复制的资源并不是立即被复制，而是与原来的对象共享相同的内存，将复制推迟到写入时才进行。这样，如果没有写入操作，就不会产生内存复制，从而大大提高效率。

对于一个字符串来说，同样可以使用 COW 技术。当一个字符串对象复制给另一个字符串对象的时候，底层保存了一堆字符的内存并不发生复制，只有对字符串进行修改时才将其复制一份出来以便使得修改操作不影响原字符串。

请尝试给上一题中的 `MyString` 类增加 COW 技术，使得其拷贝构造函数不会复制内存（不需要为 `append()` 实现 COW）。当然，仍然不允许有内存泄漏或内存重复释放。

注意：从 `char *` 构造 `MyString` 时即使有 COW 内存也必须被复制（想一想为什么？）

若你的 COW 实现正确，那么在第三题给出的测试样例中，内存复制会发生在此处：

```
    mystr3[1] = ' ';
// ^^^^^^^^^^^^^^^^   There should be memory copying here
    cout << mystr3 << endl;         // a string
```

但紧随其后的修改操作则无需再次复制内存：

```
    mystr3[1] = ' ';
    cout << mystr3 << endl;         // a string
```

```
   mystr3[2] = 'b';
// ^^^^^^^^^^^^^^^^   There should be no memory copying here
   cout << mystr3 << endl;         // a btring
```