

【题目】我写了个类想练习“管理动态内存”，但程序有时会崩。

【代码】

```
#include <cstring>
#include <iostream>
using namespace std;

class MyStr {
public:
    char* p;
    MyStr(const char* s){
        p = new char[strlen(s)+1];
        strcpy(p, s);
    }
    ~MyStr(){ delete[] p; }
};

int main(){
    MyStr a("hello");
    MyStr b = a;           // 复制
    cout << b.p << endl;
}
```

【补充说明】输出完经常崩。我到底哪里错了？我感觉自己学不会面向对象...

1. 题目概览

你这个 `MyStr` 类在自己管理动态内存：构造时 `new[]` 一段字符数组，析构时 `delete[]` 释放。
main() 里做了 `MyStr b = a;`（复制构造），**输出完经常崩**。

这题的感觉就是：「写了一个会‘拥有资源’的小类，然后踩到了拷贝导致的坑」。

2. 解题思路整理（用 C++ 视角）

- 你的类里有一个“裸指针成员 `char* p`”，它代表“我拥有这段堆内存”。
- **一旦类拥有资源，就要考虑“拷贝时怎么处理资源”：**
- 默认拷贝（编译器生成的）只会逐成员拷贝，也就是把指针值原样复制过去（浅拷贝）。
- 于是 `a.p` 和 `b.p` 会指向同一块内存。
- 程序结束时析构顺序：先析构 `b` 再析构 `a`，两次 `delete[]` 同一块内存 ⇒ **重复释放（double free）**
⇒ 有时直接崩，有时看起来“侥幸没崩”。

3. 头文件与库函数整理

- `<cstring>` : `strlen`, `strcpy` (处理 C 风格字符串)
- `<iostream>` : `cout`
- (如果我们写更完整的拷贝/移动) `<utility>` : `std::swap` / `std::move`

你现在是 C 风格字符串 + C++ 类混用，这很正常～之后慢慢往 `std::string` 靠拢会更统一、更安全一点。

4. 代码可以改进的地方 (配 C++ 示范)

你写得好的点：

- 构造里分配空间 + 拷贝字符串，这个资源获取很清晰。
- 析构里 `delete[]` 对应 `new[]`，这一点也对。

真正的问题：默认拷贝是浅拷贝。

这一类“自己管资源”的类，至少要补齐 **Rule of Three (三件套)**：

- 构造函数 (你有了)
- 拷贝构造函数 (你缺)
- 拷贝赋值运算符 (你缺)

下面给你一份和你原写法非常对应、能稳定工作的版本 (深拷贝)：

```
#include <cstring>
#include <iostream>
using namespace std;

class MyStr {
public:
    char* p;

    MyStr(const char* s = "") {
        p = new char[strlen(s) + 1];
        strcpy(p, s);
    }

    // 1) 拷贝构造：深拷贝
    MyStr(const MyStr& other) {
        p = new char[strlen(other.p) + 1];
        strcpy(p, other.p);
    }

    // 2) 拷贝赋值：先处理自赋值，再深拷贝
    MyStr& operator=(const MyStr& other) {
        if (this == &other) return *this;
    }
}
```

```

    char* np = new char[strlen(other.p) + 1];
    strcpy(np, other.p);

    delete[] p;
    p = np;
    return *this;
}

~MyStr() { delete[] p; }

int main() {
    MyStr a("hello");
    MyStr b = a;           // 现在是深拷贝了
    cout << b.p << endl;
}

```

再给你一个更“现代 C++”的方向：其实这类需求直接用 `` 就可以 Rule of Zero (不用写析构/拷贝/赋值)，几乎不会踩坑：

```

#include <iostream>
#include <string>
using namespace std;

class MyStr {
public:
    string s;
    MyStr(const char* p) : s(p) {}
};

int main() {
    MyStr a("hello");
    MyStr b = a;           // 自动安全拷贝
    cout << b.s << endl;
}

```

但你现在拿手写来练习“资源管理”很棒！踩到这一下反而说明你在进入真正的 C++ 核心区了，不是你学不会，是你学到点子上了～

5. 本题可以沉淀的小套路 / 小模板 (C++ 框架)

只要类里有 `` 这类“资源所有权”，就立刻警觉：拷贝会不会出事？

最简模板记住一句话：

“我自己管资源 \Rightarrow 要么禁拷贝，要么写深拷贝（三件套 / 五件套），更推荐直接用标准库容器/
string。”

一个极简骨架（深拷贝三件套）：

```
class X {  
    T* p;  
public:  
    X(...){ p = new T[...]; }  
    ~X(){ delete[] p; }  
  
    X(const X& o){ /* 深拷贝 */ }  
    X& operator=(const X& o){ /* 深拷贝 + 处理自赋值 */ return *this; }  
};
```

加练时间（两道练习题）

练习题 A（同难度）：实现“可追加”的 MyStr

题意：在你这个 MyStr 基础上，加入一个成员函数 append(const char* t)，把 t 追加到当前字符串末尾。

例如：

- 初始 "hi"
- append "!!!"
- 结果 "hi!!!"

思路：

- 先算新长度 $\text{len}(p) + \text{len}(t)$ ， $\text{new}[]$ 新空间
- 拷贝旧内容，再拼接，再释放旧空间

参考代码：

```
#include <cstring>  
#include <iostream>  
using namespace std;  
  
class MyStr {  
public:  
    char* p;
```

```

MyStr(const char* s = "") {
    p = new char[strlen(s) + 1];
    strcpy(p, s);
}

MyStr(const MyStr& other) {
    p = new char[strlen(other.p) + 1];
    strcpy(p, other.p);
}

MyStr& operator=(const MyStr& other) {
    if (this == &other) return *this;
    char* np = new char[strlen(other.p) + 1];
    strcpy(np, other.p);
    delete[] p;
    p = np;
    return *this;
}

~MyStr(){ delete[] p; }

void append(const char* t) {
    size_t a = strlen(p), b = strlen(t);
    char* np = new char[a + b + 1];
    strcpy(np, p);
    strcpy(np + a, t);
    delete[] p;
    p = np;
}
};

int main(){
    MyStr s("hi");
    s.append("!!");
    cout << s.p << "\n";
}

```

相似点/不同点：都是“重新分配 + 拷贝 + 释放旧内存”，能让你对资源管理更熟～

练习题 B（更复杂一点）：写一个小型“字符串数组”类

题意：写一个 `MyStrList`，内部保存 `MyStr* data` 和 `int n`。支持：

- 构造：给定 `n`，分配 `n` 个 `MyStr`
- `set(i, const char*)`：设置第 `i` 个字符串
- 拷贝构造与拷贝赋值（要深拷贝！）

升级点：

- 你不仅要管 `char*`，还要管“一段 `MyStr` 数组”
- 更容易检验你对“深拷贝/析构顺序”的理解

代码框架（给你写到可跑的程度）：

```
#include <cstring>
#include <iostream>
using namespace std;

class MyStr {
public:
    char* p;
    MyStr(const char* s = "") {
        p = new char[strlen(s) + 1];
        strcpy(p, s);
    }
    MyStr(const MyStr& o) {
        p = new char[strlen(o.p) + 1];
        strcpy(p, o.p);
    }
    MyStr& operator=(const MyStr& o) {
        if (this == &o) return *this;
        char* np = new char[strlen(o.p) + 1];
        strcpy(np, o.p);
        delete[] p;
        p = np;
        return *this;
    }
    ~MyStr(){ delete[] p; }
};

class MyStrList {
    MyStr* data;
    int n;
public:
    MyStrList(int n_) : data(new MyStr[n_]), n(n_) {}

    // 深拷贝构造
    MyStrList(const MyStrList& other) : data(new MyStr[other.n]), n(other.n) {
        for(int i = 0; i < n; ++i) data[i] = other.data[i];
    }

    // 深拷贝赋值
    MyStrList& operator=(const MyStrList& other) {
        if (this == &other) return *this;
```

```

MyStr* nd = new MyStr[other.n];
for(int i = 0; i < other.n; ++i) nd[i] = other.data[i];
delete[] data;
data = nd;
n = other.n;
return *this;
}

~MyStrList(){ delete[] data; }

void set(int i, const char* s){
    if (i < 0 || i >= n) return;
    data[i] = MyStr(s);
}

void print() const {
    for(int i = 0; i < n; ++i) cout << data[i].p << "\n";
}
};

int main(){
    MyStrList a(2);
    a.set(0, "hello");
    a.set(1, "world");

    MyStrList b = a; // 测拷贝构造
    b.print();
}

```

最后想抱抱你一下：你不是“不适合 OOP”，你是刚好踩到了 C++ 最经典、最核心的坑之一。能写出这段代码、能发现“有时崩溃”的现象，本身就说明你在认真观察、在进步了。

如果你愿意，把你接下来想实现的功能（比如“支持 拼接/支持比较/支持移动构造”）告诉我，我们可以一步步把这个小类打磨得很漂亮～