很好。既然你要求加大难度并进行“综合性复习”，那我们就把 **Day 4（内存管理）** 的幽灵请回来，让它和 **Day 7（Const 契约）** 联手给你制造一点麻烦。

今天的题目不仅考查语法，更考查你的**类设计哲学**。

### Day 7 挑战：固件完整性验证器 (Firmware Integrity Verifier)

背景：

你正在设计一个安全芯片的 ROM 控制器 (SecureROM)。

ROM 的物理特性决定了：

1. **不可篡改**：一旦烧录（构造），内容绝对不能变。
2. **资源独占**：每个 ROM 模块对应物理内存的一块独立区域，不能有两块芯片共用同一个物理地址（防篡改）。
3. **只读接口**：外部只能读取数据或计算校验和，绝对不能通过后门写入。

### 规格说明书 (Spec Sheet)

#### 1. 类定义 SecureROM

你需要在堆（Heap）上管理一块裸内存。

* **私有成员**：
  + unsigned char\* m\_Data: 指向堆内存的裸指针。
  + int m\_Size: 数据大小。
* **构造函数 SecureROM(int size)**：
  + 在堆上分配 size 大小的内存。
  + **复习点**：用 0x00 初始化所有字节（使用 memset 或循环）。
  + 为了模拟烧录，把第 0 个字节设为 0xAA，最后一个字节设为 0xBB。
* **析构函数**：
  + **复习点**：确保没有内存泄漏。

#### 2. Const 契约（核心考点）

外部世界需要与你的 ROM 交互，但你必须限制他们的权限。请实现以下成员函数，并**正确加上 const 修饰符**：

* unsigned char ReadByte(int index)：返回指定位置的字节。
  + *要求*：这个函数必须能在 const 对象上调用。
* int CalculateChecksum()：返回所有字节的累加和。
  + *要求*：这个函数绝不能修改 m\_Data 里的任何内容。

#### 3. 这里的陷阱：拷贝构造函数 (The Copy Trap)

这是今天的高阶复习点。

在 main 函数中，我会让你把 SecureROM 对象按值传递 (Pass by Value) 给一个验证函数。

* **默认行为**：C++ 默认的拷贝是“浅拷贝”（Shallow Copy）。它只复制指针的值，不复制指针指向的内存。
* **后果**：当函数结束，局部对象的析构函数会释放内存。main 函数里的原对象再析构时，会试图释放同一块内存 —— **Double Free Error（双重释放），程序直接崩溃。**
* **任务**：你需要手动编写一个**拷贝构造函数 (Copy Constructor)**，实现“深拷贝”（Deep Copy）。
  + SecureROM(const SecureROM& other)
  + 你必须申请一块**新**的内存，把 other 的数据完完整整复制过来。

### 你的任务代码结构

请在 main.cpp 中实现以下逻辑：

C++

#**include** <iostream>  
#**include** <cstring> // for memset or memcpy  
  
class SecureROM {  
private:  
 unsigned char\* m\_Data;  
 int m\_Size;  
  
public:  
 // 1. 构造函数  
 SecureROM(int size) {  
 // TODO: 分配内存，初始化  
 std::cout << "Allocated " << size << " bytes at " << (void\*)m\_Data << std::endl;  
 }  
  
 // 2. 析构函数  
 ~SecureROM() {  
 // TODO: 释放内存  
 std::cout << "Freed memory at " << (void\*)m\_Data << std::endl;  
 }  
  
 // 3. 自定义拷贝构造函数 (Deep Copy)  
 SecureROM(const SecureROM& other) {  
 std::cout << "Copying ROM..." << std::endl;  
 // TODO: 申请新内存，复制数据  
 // 提示：m\_Size = other.m\_Size; m\_Data = new ...  
 }  
  
 // 4. Const 成员函数  
 // TODO: 实现 ReadByte 和 CalculateChecksum  
 // 注意：必须加上 const 关键字  
};  
  
// 外部工具函数：按值传递，故意触发拷贝构造  
void VerifyFirmware(SecureROM rom) {  
 std::cout << "Verifying Checksum: " << rom.CalculateChecksum() << std::endl;  
}  
  
int main() {  
 // 1. 创建一个 ROM  
 SecureROM masterChip(1024);  
  
 // 2. 模拟指针权限控制 (Const Pointers)  
 // 任务：定义一个指针 pReadOnly，指向 masterChip。  
 // 要求：可以通过 pReadOnly 调用 ReadByte，但绝不能通过它修改 masterChip (虽然 SecureROM 本身也没提供修改接口，但要在语法层面锁死)。  
 const SecureROM\* pReadOnly = &masterChip;  
  
 // 3. 触发深拷贝逻辑  
 std::cout << "--- Starting Verification ---" << std::endl;  
 VerifyFirmware(masterChip); // 这里会调用拷贝构造函数  
 std::cout << "--- Verification Done ---" << std::endl;  
  
 // 4. 程序结束时，masterChip 析构。  
 // 如果你的拷贝构造函数写错了，这里会崩溃 (Double Free)。  
 return 0;  
}

### 你的挑战清单

1. **内存**：正确使用 new 和 delete[]。
2. **Const**：ReadByte 和 CalculateChecksum 如果不加 const，VerifyFirmware 里的 rom.CalculateChecksum() 可能会报错吗？为什么？
3. **深拷贝**：如果你注释掉你写的拷贝构造函数，观察程序运行结果（崩溃或乱码）。加上后，观察“Allocated”打印了多少次？（应该打印 2 次：一次是 masterChip，一次是复制品）。

**开始编码。这是一个真正合格的 C++ 工程师必须迈过的坎：The Rule of Three (构造、析构、拷贝构造)。**