很好。既然你已经准备好面对 C++ 最令人敬畏的“指针”了，那我们就直接进入深水区。

这一关的目的是让你理解：**指针就是遥控器，它允许你隔空操纵内存。**

### 题目：死亡判定与空指针防御 (Pointer Manipulation & Safety)

背景：

在游戏开发中，我们经常需要把核心数据（如玩家血量、得分）放在一个中心位置（比如 main 函数或 Game 类），然后让各种子系统（如战斗系统、陷阱系统）去修改它。为了避免昂贵的数据拷贝，我们传递地址。

**需求说明：**

1. **编写 TakeDamage 函数**
   * **函数签名**：void TakeDamage(int\* pHealth, int\* pScore, int damage)
   * **参数**：
     + pHealth：指向玩家血量的指针。
     + pScore：指向玩家分数的指针。
     + damage：受到的伤害值（整数）。
   * **功能逻辑**：
     + **第一道防线 (Safety)**：如果传入的 pHealth 或 pScore 中有任何一个是空指针 (nullptr)，函数必须**立即返回**，不做任何操作。*(这是工业级代码的基本素养)*
     + **核心逻辑**：通过指针扣除 pHealth 指向的血量。
     + **连锁反应**：扣血后，检查血量。如果血量**小于等于 0**，并且**还没有死透**（这一步逻辑你自己思考怎么写才合理，或者简单点：只要血量 <= 0，就惩罚分数），将 pScore 指向的分数扣除 50 分。
     + **注意**：不要在函数里打印任何东西。
2. **编写 main 函数**
   * 定义两个整数变量：health 初始值为 100，score 初始值为 100。
   * **测试 1 (正常伤害)**：调用 TakeDamage，传入 health 和 score 的地址，造成 20 点伤害。
     + 打印此时的 health 和 score（预期：80, 100）。
   * **测试 2 (致死打击)**：再次调用 TakeDamage，造成 90 点伤害。
     + 打印此时的 health 和 score（预期：-10, 50）。
   * **测试 3 (空指针攻击)**：调用 TakeDamage，传入 nullptr 作为血量指针，任意伤害值。
     + 打印 "Tried nullptr call"。
     + 程序**绝不能崩溃**。

**陷阱预警：**

* 你现在的工具是 \*（解引用）和 &（取地址）。搞混一个，程序就崩。
* 操作指针前不检查 nullptr 是 C++ 程序员被解雇的常见原因之一。

**现在，写出完整的代码。** 证明你能驾驭内存地址，而不是被它绊倒。