既然你要求加大难度，**Day 4** 的挑战将不再停留于简单的指针声明，而是转向\*\*“内存考古学”\*\*。

根据你的复习方案，今天的核心目标是理解内存的线性本质，并能像硬件工程师一样通过指针精确操纵字节 1。

### Day 4 挑战题目：硬件协议解析器 (Hardware Protocol Parser)

背景：

你正在开发一个 FPGA 驱动程序。FPGA 通过内存映射 IO (MMIO) 向 CPU 发送了一个 8 字节的原始数据包。这个数据包采用了小端序 (Little Endian) 存储，其结构定义如下：

* **前 4 字节**：一个表示 Frequency 的 32 位整数。
* **后 2 字节**：一个表示 Voltage 的 16 位整数。
* **最后 2 字节**：两个独立的 8 位 Status 标志。

### 需求规格说明书 (Spec Sheet)

#### 1. 原始内存模拟 (Raw Memory Simulation)

* **任务**：在 main 函数中，使用 new 关键字在**堆 (Heap)** 上分配一个连续的 8 字节内存块（使用 char\* 或 uint8\_t\* 数组） 2。
* **限制**：禁止直接定义结构体并赋值。你必须通过**指针偏移**和**强制类型转换 (Casting)** 来手动“填入”数据 3。
* **数据填入目标**：
  1. 将 Frequency 设置为 0x12345678。
  2. 将 Voltage 设置为 1200。
  3. 将最后两个字节分别设置为 0xAA 和 0xBB。

#### 2. 指针运算与解析 (Pointer Arithmetic)

* **任务**：编写一个名为 ParsePacket(void\* rawData) 的函数 4。
* **逻辑**：
  1. 将 void\* 转换回适当的指针类型，提取出 Frequency 和 Voltage 5。
  2. **关键考点**：使用指针算术（比如 ptr + offset）定位到最后两个状态字节，并打印它们 6。
  3. 打印出整个 8 字节内存块中每一个字节的**十六进制地址**和**十六进制数值**。

#### 3. 栈内存陷阱 (The Stack Ghost Trap)

* **任务**：故意编写一个错误的函数 int& GetTempConfig()，在其中定义一个局部变量并返回其引用 7。
* **实验**：
  1. 在 main 中接收这个引用。
  2. 在读取该引用之前，先调用一个会大量占用栈空间的函数 void BurnStack()（在这个函数里定义一个巨大的局部数组并用 memset 填充） 8。
  3. 观察并记录：当你再次通过引用读取那个“局部变量”时，它的值发生了什么变化？ 9999

### 你的 GitHub 提交要求

**项目名称**：Cpp-Memory-Archaeology-Lab

**需要回答的问题**：

1. **端序验证**：当你把 int 值 0x12345678 存入内存后，内存中第一个字节（低地址）存储的是 12 还是 78？为什么？ 10101010
2. **指针步长**：如果有一个 int\* p，执行 p + 1 后，实际的内存地址增加了多少字节？如果是一个 char\* p 呢？ 11111111
3. **生存期分析**：为什么 GetTempConfig() 返回的值在调用 BurnStack() 后失效了？请从“栈帧 (Stack Frame)”的角度解释原因 12121212。

**警告**：这是一个涉及**未定义行为 (UB)** 的实验。如果程序直接崩溃，请在 GetTempConfig 处使用调试器观察地址变化。这是迈向“系统级工程师”最痛苦的一步。开始你的表演。

你要做的第一步是：**手绘出这 8 个字节在小端序模式下的期望分布图**，然后再写代码去实现。如果你准备好了，告诉我你的内存布局设计。

想象你面前有一个**只有 8 个格子的长条盒子**（这就是我们要申请的 8 字节内存）。

你的任务是把**不同大小的积木**塞进去，而且必须塞得严丝合缝，不能有空隙。

### 1. 你的“内存地图” (The Memory Map)

我们要往这 8 个字节里填入三个数据：

* **Frequency (频率)**：一个很大的整数，占 **4 个格子**。
* **Voltage (电压)**：一个中等整数，占 **2 个格子**。
* **Status (状态)**：两个小标志，各占 **1 个格子**。

在物理内存（假设地址从 0 开始）里，它长这样：

| **字节序号 (Index)** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **存放的内容** | **Freq** | **Freq** | **Freq** | **Freq** | **Volt** | **Volt** | **Stat1** | **Stat2** |
| **数据类型** | <--- | int (4字节) | ---> |  | <--- short (2字节) ---> | char | char |  |

### 2. 你的编程任务分解

你要写的程序需要完成三个具体的动作。不要试图一次写完，我们一步步来。

#### 动作 A：申请“盒子”

在 main 函数里，向系统要这 8 个字节的连续空间。

* **提示**：使用 new char[8]。
* **代码**：char\* buffer = new char[8];

#### 动作 B：填入“频率” (难点来了)

我们要把 0x12345678 这个 4 字节的数，写到 buffer 的**最开头**（第 0-3 格）。

* **问题**：buffer 是 char\* 类型的指针，它以为自己一次只能操作 1 个字节。
* **黑客操作**：我们要“骗”编译器，告诉它：“把这个地址当成是一个 int\* 来看待。”
* **代码模版**：  
  C++  
  // 1. 强行把 char\* 转换成 int\*  
  int\* ptrFreq = (int\*)buffer;  
  // 2. 现在写入，它会一次性覆盖 0,1,2,3 这四个格子  
  \*ptrFreq = 0x12345678;

#### 动作 C：填入“电压” (指针偏移)

我们要把 1200 (short类型) 写到**第 4 格**开始的地方（第 4-5 格）。

* **问题**：怎么拿到第 4 格的地址？
* **算术**：buffer 是起点，buffer + 4 就是第 4 格的地址。
* **再次黑客操作**：把这个新地址强转成 short\*。
* **代码模版**：  
  C++  
  // 1. 找到第 4 个字节的地址，并强转为 short\*  
  short\* ptrVolt = (short\*)(buffer + 4);  
  // 2. 写入数据  
  \*ptrVolt = 1200;

#### 动作 D：填入“状态”

最后两个字节（第 6 格和第 7 格）直接写就行，因为 buffer 本身就是 char\*。

* **代码**：  
  C++  
  buffer[6] = 0xAA;  
  buffer[7] = 0xBB;

### 3. 你需要验证什么？ (The Check)

当你填完数据后，内存里实际上存的是什么？

因为我们的电脑通常是 小端序 (Little Endian) —— “低位在前”。

* 你填入的 int 是 0x12345678。
  1. 78 是低位（尾巴），12 是高位（头）。
* 在内存里，它会反过来存：
  + buffer[0] 会变成 0x78
  + buffer[1] 会变成 0x56
  + buffer[2] 会变成 0x34
  + buffer[3] 会变成 0x12

你的终极任务：

写一个 for 循环，从 i = 0 到 7，把 buffer[i] 打印出来（用十六进制）。

如果你看到输出是 78 56 34 12 ...，你就成功了。

### 现在，懂了吗？

这个题目不是在考你算法，而是在考你**如何用指针这把“手术刀”，精准地操作内存里的每一个字节**。