C++ `for` 循环(二) 讲义

1. 引言

问题引入

在上一节课，我们学习了 `for` 循环，并用它解决了“重复打印N次”的问题，我们知道 `for` 循环是一个强大的**计数器**。但如果只是用它来数数，那可就太小看它了！

现在，我们来看另一个问题：

老师现在要大家一个一个地报上自己的学号，但我不知道咱们班有多少同学。在我听完所有人的学号后，我需要立刻说出那个**最小**的学号。

作为老师，我应该怎么做？需要把几十个学号都记在脑子里吗？太难了！我们的大脑会自动采用一种更聪明的策略：只记住“到目前为止听到的那个最小学号”，然后用新报上来的学号和这个“记录”进行比较。

价值阐述

今天我们将要学习的，是 `for` 循环在算法中最常见的三种应用模式：

1. **打擂台模式**：用于在一堆数据中寻找“最”值（最大、最小、最长、最强...）。
2. **累加模式**：用于对一连串数据进行求和、求积等累积性计算。
3. **分类统计模式**：用于遍历数据，并根据不同条件将它们归类、分别统计。

掌握了这三大模式，就能解决诸如数据分析、统计、模拟等一大批实际问题。

---

2. 核心思想与模式

模式一：打擂台算法 —— 寻找“最”值

这是解决“求最大/最小值”问题的模板。让我们用书本上**例4-2** “找最小值”的例子，来学习这个算法。

核心思想：**只记住当前最优值，然后不断与新的挑战者进行比较**。

步骤拆解：

1. **【搭建擂台】**
   - **比喻**：比赛开始前，我们需要一个擂主。为了保证第一个挑战者一定能赢，我们故意请一个最弱的虚拟擂主站在台上。
   - **代码实现**：我们需要一个变量 `min_val` 来扮演“擂主”。因为我们要找最小值（越小越强），所以最弱就意味着一个**非常大**的数。
   - `int min_val = 100000000;` // 一个足够大的初始值
2. **【挑战者依次上台】**
   - **比喻**：n个选手排好队，一个接一个上台挑战。
   - **代码实现**：我们用 `for` 循环n次，每次在循环体内读入一个新的“挑战者”。
   - `for (int i = 0; i < n; i++) { cin >> tmp; ... }`
3. **【判断胜负，更新擂主】**
   - **比喻**：如果挑战者比擂主“更强”（在这里是数值更小），他就挑战成功，成为新擂主。否则，擂主守擂成功。

- **代码实现**：在循环体内，用一个 `if` 语句来判断。
- `if (tmp < min_val) { min_val = tmp; }`

4. **【比赛结束，留在最后的就是冠军】**
   - **比喻**：所有人都挑战完毕后，最后还站在台上的，就是真正的冠军。
   - **代码实现**：当 `for` 循环结束后，`min_val` 变量里存储的值，就是我们要找的最小值。直接输出它即可。

模式二：累加模式 —— 求和

核心思想：准备一个容器变量，在循环中不断将自身和新的东西加在一起，最终越滚越大。

步骤拆解（往存钱罐里放硬币）：

1. **【准备一个空的存钱罐】**
   - **比喻**：在开始存钱之前，你必须有一个**空**的存钱罐。
   - **代码实现**：定义一个求和变量 `sum`，并**必须**将它初始化为 `0`。这个操作**必须在循环外**完成。
   - `int sum = 0;`
2. **【确定要放哪些硬币】**
   - **比喻**：我们要依次放入面值为 1, 2, 3, ..., 100 的硬币。
   - **代码实现**：用 `for` 循环来生成 1 到 100 这一串数字。
   - `for (int i = 1; i <= 100; i++) { ... }`
3. **【放硬币】**
   - **比喻**：把当前这枚硬币 `i` 丢进存钱罐 `sum`。
   - **代码实现**：存钱罐里新的总数，等于它原来的数，加上当前硬币的面值。
   - `sum = sum + i;` (或简写为 `sum += i;`)

关键：求和变量 `sum` 的初始化**必须**在循环外部。如果放在循环内部，它每一轮都会被"清空"回0，那永远也攒不下钱！

---

3. 分类统计模式(重要)

例如想象一个大型的快递分拣中心需要分类统计：

- 有一个**传送带**，上面源源不断地运来包裹。
- 传送带旁边站着一个**分拣员**，他的任务是看每个包裹的目的地，然后把它放到对应的城市货车里（比如北京、上海、广州）。
- 当所有包裹都处理完后，我们就可以很轻松的统计出每辆对应的货车里装了多少包裹。

在编程中，这个模式的逻辑是完全一样的，我们通过书本上实际的案例进行讲解。

案例：例4-3 分类求平均数

问题：从1到n这些数中，是k的倍数分为A类，不是的分为B类。求A、B两类数各自的平均数。

分析：`for` 循环像一个**传送带**，负责把1到n的数字一个一个送过来；循环内部的 `if/else` 像一个**分拣员**，判断这个数字该去A队还是B队。

总结起来就是：**分类统计模式中 `for` 循环负责遍历所有的数据，而 `if/else` 判断负责对这些数据挨个进行分类并依照规则进行处理**。

分步演示

1. **准备工作（循环外）**
   - 我们需要统计A、B两队的“总分”和“人数”，才能求平均分。所以，我们需要4个“存钱罐”！
   - `double Asum = 0, Bsum = 0;`
   - `int Acount = 0, Bcount = 0;`
2. **循环**
   - 我们需要遍历从1到n的所有数字。
   - `for (int i = 1; i <= n; i++) { ... }`
3. **分类进行处理**
   - 每从传送带上拿到一个数字 `i`，就要判断它是不是k的倍数（`i % k == 0`）。
   - 使用 `if-else` 结构：
     - 如果为真，是A队成员。就把 `i` 累加到 `Asum`，并让 `Acount` 加1。
     - 否则，是B队成员。就把 `i` 累加到 `Bsum`，并让 `Bcount` 加1。
4. **计算结果（循环后）**
   - 当循环结束，我们就得到了两队各自的总分和人数。
   - 分别计算平均分 `Asum / Acount` 和 `Bsum / Bcount` 并输出。

---

4. 代码实现

找最小值（打擂台）

```
int main() {
    int n;
    cin >> n; // 输入数字的总个数

    int min_val = 100000000; // 储存擂主，初始化为极大值，确保第一个人能打败它
    int tmp; // 用于存放当前的挑战者

    for (int i = 0; i < n; i++) {
        cin >> tmp; // 读入一个挑战者

        //  如果挑战者更小，就更新擂主
        if (tmp < min_val) {
            min_val = tmp;
        }
    }

    // 循环结束，输出最终的擂主
    cout << min_val << endl;
    return 0;
}
```

求1到100的和（累加）

```
int main() {
    // 准备一个空的存钱罐
    int sum = 0;

    // 循环生成 1 到 100 的硬币
    for (int i = 1; i <= 100; i++) {
        // 把当前的硬币 i，放入存钱罐 sum
        sum += i;
    }

    // 循环结束后，输出存钱罐里的总数
    cout << sum;
    return 0;
}
```

---

5. 深化

变量的作用域

这是一个非常重要的概念！

- **循环内的临时工**：在 `for (int i = 0; ...)` 中定义的变量 `i`，它的“生命周期”仅限于这个 `for` 循环。一旦循环结束，变量 `i` 就会消失，在循环外部无法使用它。
- **循环外的长期工：像 `sum`，`min_val`，`Acount` 这些需要在循环结束后仍然保留其计算结果的变量，必须在循环外部定义。**

工具箱：`min()` 和 `max()` 函数

对于打擂台算法中的找到两个数中的最小值，C++提供了一个更便捷的工具。`min(a, b)` 函数会直接返回a和b中较小的那一个。

- **使用前提**：需要包含头文件 `#include <algorithm>`。
- **简化代码**：打擂台算法中的核心判断 `if (tmp < min_val) { min_val = tmp; }` 可以被简化为一句话：
  `min_val = min(min_val, tmp);`
- 这句话的意思是：“新的擂主将会被赋值为‘旧擂主’和‘挑战者’中更小的那一个”，逻辑等价，但代码更简洁。

---

6. 总结

- **打擂台模式**：`初始化最值 -> 循环遍历打擂台 -> if判断更新`，用于求最大/最小值。
- **累加模式**：`初始化累加器=0 -> 循环遍历 -> sum += i`，用于求和。
- **分类统计模式**：`for` 循环提供数据，`if-else` 进行分类处理，而累加器分别统计不同的分类数量。
- **变量定义位置**：准备工作（初始化）在**循环外**，重复性处理在**循环内**。