# C++ `for` 循环(一) 讲义

## 1. 引言

### 问题引入

我们来看一个非常经典的题目：**在屏幕上打印从 1 到 10 的所有整数。**

用我们学过的 `while` 循环，可以这样实现：

```
int main() {
    int i = 1;          // 1. 循环变量初始化（起点）
    while (i <= 10) {   // 2. 循环条件判断（终点）
        cout << i << endl;
        i++;            // 3. 循环变量更新（推进）
    }
    return 0;
}
```

这段代码完全正确，但仔细观察一下：为了完成这个循环，我们定义了三个关键部分：**初始化 `i`**、**判断 `i`**、**更新 `i`**。但这三个兄弟却被分散在了代码的三个不同位置。如果循环体很长，更新语句 `i++` 可能会离 `while` 很远，一不小心就容易忘记写，从而导致无限循环！

**思考一下**：有没有一种方法，能把这三个控制循环的核心要素“捆绑”在一起，让我们的意图一目了然呢？

### 价值阐述

C++ 的设计者们也注意到了这个问题。对于这种**“次数确定”**或**“范围已知”**的循环（比如循环10次，从1循环到100等），他们专门设计了一种语法结构，将循环的“三要素”集中管理。这就是 `for` 循环。

---

## 2. 核心思想

### 思路铺垫

`while` 循环的逻辑是分散的，而 `for` 循环的核心思想就是**“集中管理”**。它让我们把所有控制循环进程的操作都放在一起。

### `for` 循环的语法

让我们来看看 `for` 循环的语法长什么样（非常重要！）：

```
for (① 循环变量初始化; ② 循环条件判断; ③ 循环变量更新) {
    // 循环体...
}
```

它由三个部分组成，用**分号** `;` 隔开：

1. `① 循环变量初始化`
    - **作用**：设置循环的**起点**。比如 `int i = 1;`。
    - **执行时机**：**只在**整个 `for` 循环开始时执行**唯一一次**。
2. `② 循环条件判断`

- **作用**：决定循环**是否继续**。这是一个布尔表达式，比如 `i <= 10;`。
- **执行时机**：**每一次**循环体执行**之前**，都会进行判断。如果为 `true`，就执行循环体；如果为 `false`，循环立即结束。

3. `③ 循环变量更新`
   - **作用**：让循环**向前推进**，确保循环最终能够结束。比如 `i++`。
   - **执行时机**：**每一次**循环体执行**之后**，都会执行这个更新操作。

## 步骤拆解：`for` 循环是如何运行的？

我们可以把 `for` 循环想象成在操场上跑步：

`for (int i= 1; i<= 5; i++)`

1. **【准备开跑】(执行 ① 初始化)**
   - `int i= 1;` (我准备从第1圈开始跑)
   - 这个动作只在最开始做一次。
2. **【检查】(执行 ② 条件判断)**
   - `i<= 5?` (我跑完的圈数是否小于等于5圈？)
   - **当前是第1圈**：`1 <= 5` -> `true`。好的，可以跑！
3. **【执行任务】(执行循环体)**
   - `{ ... }` (跑一圈！)
4. **【更新状态】(执行 ③ 更新)**
   - `i++` (圈数加一，我现在跑完1圈，准备跑第2圈了)
   - `i` 的值现在变成了 `2`。
5. **【回到第2步，开始新一轮】**
   - **跑前检查**：`i<= 5?` (当前是第2圈) -> `2 <= 5` -> `true` -> 继续跑！
   - **执行任务**：跑一圈！
   - **更新状态**：`i++` -> `i` 变成 `3`。
   - ...这个过程不断重复...
6. **【循环终点】**
   - ...当 `i` 变成 `5` 时，检查 `5 <= 5` 依然为 `true`，跑完第5圈。
   - **更新状态**：`i++` -> `i` 变成 `6`。
   - **跑前检查**：`i<= 5?` (当前是第6圈) -> `6 <= 5` -> `false`。
   - **条件不成立，循环立即结束！** 程序跳出 `for` 循环，执行后面的代码。

**执行流程**：`① -> (② -> 循环体 -> ③) -> (② -> 循环体 -> ③) -> ... -> ② (条件为false) -> 结束`

---

## 3. 实例

### 案例：打印1到5

用刚才学到的知识，写出通过 `for` 循环打印输出1到5的代码。

```
for (int i = 1; i <= 5; i++) {
    cout << i << endl;
}
```

### 分步演示

| 轮次 | 检查 `i <= 5`？ | `i` 的值 | 循环体操作 (输出) | 更新 `i++` 后，`i` 变为 |
|---|---|---|---|---|
| **开始** | - | `1` (来自初始化) | - | - |
| **第1轮** | `1 <= 5` (True) | 1 | 输出 `1` | `2` |
| **第2轮** | `2 <= 5` (True) | 2 | 输出 `2` | `3` |
| **第3轮** | `3 <= 5` (True) | 3 | 输出 `3` | `4` |
| **第4轮** | `4 <= 5` (True) | 4 | 输出 `4` | `5` |
| **第5轮** | `5 <= 5` (True) | 5 | 输出 `5` | `6` |
| **第6轮** | `6 <= 5` (False) | 6 | **不执行** | **循环结束** |

---

## 4. 代码实现

现在，我们来解决书本中的 `Apples Plus` 例题。

**问题简述**：输入一个数 L，程序需要重复L次，打印 "Today, I ate x apple(s)."，其中x从1变到L。当x是1时，单词是 `apple`，否则是 `apples`。

### 分析问题

1. **重复性**：要打印L句话，这是典型的重复任务 -> **用循环。**
2. **规律性**：句子里的数字 `x` 从1递增到L -> 这和 `for` 循环的计数器 `i` 的变化一模一样！-> **确定使用 `for` 循环**，循环范围是 `1` 到 `L`。
3. **特殊性**：第1句话和后面的话格式略有不同 (`apple` 和 `apples`) -> 在循环内部需要根据当前是第几次循环（也就是 `i` 的值）来做一个**判断** -> **用 `if` 语句。**

### 完整代码

```
int main() {
    int L;
    cin >> L;

    // 搭建一个从 1 循环到 L 的 for 循环框架
    for (int i = 1; i <= L; i++) {
        // 在循环的每一次，变量 i 的值都代表当前是第几个苹果
```

```
        // 先打印每一句都相同的部分
        cout << "Today, I ate " << i << " apple";

        // 根据循环变量 i 的值，判断是否需要加 's'
        if (i > 1) {
            cout << "s";
        }

        // 打印句末的句号和换行符
        cout << "." << endl;
    }

    return 0;
}
```

### 代码解读

- `for (int i = 1; i <= L; i++)`：准备一个计数器`i`，从`1`开始，只要它没超过`L`，就要一直重复循环。每做完一次，会把`i`加一。
- `if (i > 1)`：`i`不仅仅是控制循环次数的工具，它还携带了“当前是第几次”的信息，我们可以直接用它来参与逻辑判断。

---

## 5. 常见问题与易错点

1. **无限循环**
   - **代码**: `for (int i = 1; i <= 5; i--)`
   - **错误**: 更新部分`i--`让`i`的值不断变小(1, 0, -1, ...)，导致`i <= 5`这个条件永远为真。
   - **修正**: 应改为`i++`。
2. **循环一次都不执行 (零次循环)**
   - **代码**: `for (int i = 5; i < 5; i++)`
   - **分析**: 初始化`i`为5后，第一次进行条件判断`i < 5`，即`5 < 5`，结果为`false`。因此，循环体一次都不会执行。这是`for`循环的正常行为。
3. **分号与逗号**
   - **错误**: `for (int i = 1, i <= 10, i++)`
   - **正确**: `for (int i = 1; i <= 10; i++)`
   - **切记**：`for`循环的三个部分之间必须用**分号**`;`分隔！

---

## 6. 总结

- `for`**循环**非常适合用于处理**次数确定**的循环。
- **三要素**：**初始化** (只执行1次)、**条件判断** (每次循环前)、**更新** (每次循环后)。
- **执行流程**：初始化 -> (判断 -> 循环体 -> 更新) -> (判断 -> 循环体 -> 更新) -> ...
- **循环变量**`i`它既可以是**计数器**，也是**当前循环了多少次的信息**，可以参与循环体内部的运算和判断。
- **循环嵌套分支**：在`for`循环内部使用`if`可以解决很多“普遍规律中存在特殊情况”的问题。