算术运算符与关系运算符

1. 引言

- **问题引入**：想象一下，你要编写一个游戏。角色的生命值被敌人攻击后会减少，捡到血包后会增加；你的得分会随着击败敌人而累加。这些“增加”、“减少”、“累加”的操作，在代码中如何实现呢？答案就是使用算术运算符。
- **价值阐述**：算术运算符是编程的基石。无论是简单的数值计算、复杂的科学模拟，还是游戏开发中的逻辑处理，都离不开它们。掌握好算有运算符，是写出任何有意义程序的先决条件。

2. 核心思想

- **思路铺垫**：我们从小学就开始学习数学中的“加减乘除”，C++ 中的算术运算符和它们的思想是完全一样的。它们接收一个或两个数值，然后给出一个计算结果。
- **算法精讲**：
  - **基本运算符**：`+`, `-`, `*`, `/`, `%` 这五个是我们最熟悉的。
    - `+` (加法): `5 + 2` 结果是 `7`
    - `-` (减法): `5 - 2` 结果是 `3`
    - `*` (乘法): `5 * 2` 结果是 `10`
    - `/` (除法): `5 / 2` 结果是 `2` (注意：整数除法会舍弃小数部分)
    - `%` (取模/取余): `5 % 2` 结果是 `1` (计算 `5` 除以 `2` 的余数)
  - **自增/自减运算符**：在编程中，"让一个变量加 1" 这个操作极其频繁。每次都写 `a = a + 1;` 会有点繁琐。于是，C++ 提供了一个更简洁、更高效的写法：`++` 和 `--`。
    - `++` (自增): 让变量的值加 1。`a++` 或 `++a` 都等价于 `a = a + 1`。
    - `--` (自减): 让变量的值减 1。`a--` 或 `--a` 都等价于 `a = a - 1`。
- **步骤拆解 (前置与后置的关键区别)**
  这是一个非常重要的知识点，也是初学者的一个难点。让我们把它拆解清楚：
  - **前置**: `++a` 或 `--a`
    1. **第一步：先**让变量 `a` 的值改变（加 1 或减 1）。
    2. **第二步：再**把 `a` **改变后**的新值，用于整个表达式的计算。
    - **口诀：先变，再用。**
  - **后置**: `a++` 或 `a--`
    1. **第一步：先**使用变量 `a` **原始的、未改变的**值，用于整个表达式的计算。
    2. **第二步：再**让变量 `a` 的值改变（加 1 或减 1）。
    - **口诀：先用，再变。**

3. 实例

- **具体案例**：我们通过两个简单的 `cout` 输出来看看前置和后置的差别。
- **分步演示 1：前置 `++x`**

```
int x = 5;
cout << ++x << endl; // 我们想知道这里会输出什么
```

计算机的执行过程：

1. 遇到 `++x`，这是一个前置自增。
2. **第一步 (先变)**：`x` 的值从 `5` 变为 `6`。
3. **第二步 (再用)**：使用 `x` 改变后的新值 `6` 来执行 `cout` 输出。
4. **最终结果**：屏幕上输出 `6`。

分步演示 2：后置 `y++`

```
int y = 5;
cout << y++ << endl; // 这里又会输出什么？
cout << y << endl;   // 这一行又输出什么？
```

计算机的执行过程：

1. 遇到 `y++`，这是一个后置自增。
2. **第一步 (先用)**：使用 `y` **原始的值** `5` 来执行 `cout` 输出。所以第一行会输出 `5`。
3. **第二步 (再变)**：在第一行 `cout` 语句**执行完毕后**，`y` 的值从 `5` 变为 `6`。
4. 执行第二行 `cout << y;` 时，`y` 的值已经是 `6` 了。
5. **最终结果**：屏幕上第一行输出 `5`，第二行输出 `6`。

- **复合赋值运算符**
  还有一些方便的简写形式，它们将算术运算和赋值结合起来，以下的例子中 `a` 与 `b` 均为整型变量：
  - `a += b;` 等价于 `a = a + b;`
  - `a -= b;` 等价于 `a = a - b;`
  - `a *= b;` 等价于 `a = a * b;`
  - `a /= b;` 等价于 `a = a / b;`
  - `a %= b;` 等价于 `a = a % b;`

第二部分：关系运算符

- **问题引入**：假设你要编写一个程序，判断一个学生的分数是否及格（大于等于60分）。或者，在一个游戏中，判断玩家的生命值是否已经降到了0。这些“是否大于等于”、“是否等于”的操作，单纯用算术运算符是无法完成的。我们需要新的工具来让程序进行**比较**和**判断**。
- **价值阐述**：如果说算术运算符让程序有了计算能力，那么关系运算符则赋予了程序**决策能力**。它是构建所有复杂逻辑（如 `if` 语句、循环）的基础。没有关系运算，程序就只是一台高级计算器，无法根据不同的情况做出不同的反应。
- **学习目标**：
  1. 理解并掌握 `bool` (布尔) 类型，知道它只有 `true` 和 `false` 两个值。
  2. 熟练使用 C++ 的 6 个关系运算符：`==`, `!=`, `>`, `<`, `>=`, `<=`。
  3. 理解关系运算符的运算结果是一个 `bool` 值。
  4. 掌握算术运算符和关系运算符混合使用时的**优先级**规则。

2. 核心思想

- **思路铺垫：认识一个新的数据类型 `bool` 类型**
  在学习比较之前，我们得先知道比较的结果是什么。生活中的比较，结果通常是“是”或“否”。
  - “你的年龄大于18岁吗？” -> “是” 或 “否”
  - “今天下雨了吗？” -> “是” 或 “否”
    在 C++ 中，我们用一种专门的类型来存储这种“是/否”、“真/假”的值，它就是 `bool` (布尔) 类型。
  - `true` (真): 代表“是”、“成立”。在计算机内部，通常用 `1` 来表示。
  - `false` (假): 代表“否”、“不成立”。在计算机内部，通常用 `0` 来表示。

```
bool isAdult = true;     // 定义了一个 bool 类型的变量 isAdult 的值是 true
bool isRaining = false;  // 定义了一个 bool 类型的变量 isRaining 的值是 false
```

- **关系运算符的种类**
  关系运算符就是用来比较两个值，其实很多关系与数学上是相同的。

运算符	名称	含义	例子 (a=5, b=10)	结果
`==`	等于	判断两边的值**是否相等**	`a == 5`	`true`
`!=`	不等于	判断两边的值是否不相等	`a != b`	`true`
`>`	大于	判断左边的值是否大于右边	`a > b`	`false`
`<`	小于	判断左边的值是否小于右边	`a < b`	`true`
`>=`	大于等于	判断左值是否大于或等于右值	`a >= 5`	`true`
`<=`	小于等于	判断左值是否小于或等于右值	`b <= a`	`false`

- **运算符优先级**
  当一个表达式里既有算术运算符又有关系运算符时，谁先算？比如 `3 + 4 > 2 * 3`。
  规则很简单：**先算术，后关系。**
  1. **最高级**：括号 `()`，永远最先计算。
  2. **第二级**：乘、除、取余 `*`, `/`, `%`。
  3. **第三级**：加、减 `+`, `-`。
  4. **第四级**：大小关系 `>`, `<`, `>=`, `<=`。
  5. **第五级**：相等关系 `==`, `!=`。

3. 实例

- **具体案例**：分析表达式 `3 + 4 > 2 * 3` 的计算过程。
- **分步**：
  1. **扫描表达式**：`3 + 4 > 2 * 3`
  2. **应用优先级规则**：`*` 的优先级高于 `+` 和 `>`。
     - **计算** `2 * 3`，得到 `6`。表达式变为 `3 + 4 > 6`。
  3. **再次扫描表达式**：`3 + 4 > 6`

4. **应用优先级规则**：`+` 的优先级高于 `>`。
    - **计算** `3 + 4`，得到 `7`。表达式变为 `7 > 6`。
5. **最后一步**：执行关系运算 `7 > 6`。这个比较是成立的，所以结果是 `true`。

- 而在 `cout` 中直接输出 `bool` 值时，`true` 会显示为 `1`，`false` 会显示为 `0`。所以 `cout << (3 + 4 > 2 * 3);` 会输出 `1`。

4. 代码实现

- **问题**：(书本 P33 例 3-1) 输入两个整数 `a` 和 `b`，判断并输出：
    1. `a` 是否大于 `b`？
    2. `a` 是否小于或等于 `b`？
    3. `a` 是否不等于 `b`？
       输出 3 个整数（1 代表真，0 代表假），用空格隔开。
- **完整代码**：

```
#include <iostream>
using namespace std;

int main() {
    // 定义需要输入的两个整型变量
    int a, b;

    // 从键盘读取 a 和 b 的值
    cin >> a >> b;

    // 计算第一个关系表达式 (a > b) 的结果并输出，a > b 的结果是一个 bool 值 (true 或 false)
    // cout 会自动将 true 转为 1，false 转为 0
    cout << (a > b) << " ";

    // 计算第二个关系表达式 (a <= b) 的结果并输出
    cout << (a <= b) << " ";

    // 计算第三个关系表达式 (a != b) 的结果并输出
    cout << (a != b) << endl;

    return 0;
}
```

- **代码解读**：
    - `cin >> a >> b;`：等待用户输入两个整数，分别存入变量 `a` 和 `b`。
    - `cout << (a > b) << " ";`：这一行是核心。
        - `(a > b)`：括号不是必须的，但加上能让代码更清晰。计算机会先执行括号里的 `a > b`。
        - 假如输入 `10 5`，则 `10 > 5` 结果为 `true`。
        - `cout << true;`：`cout` 会将 `true` 以整数 `1` 的形式打印到屏幕上。
        - `<< " ";`：紧接着输出一个空格，用于分隔结果。
    - 其他两行 `cout` 的逻辑完全相同。

5. 验证与深化

- **常见问题**
  1. **最大的陷阱：`==` 和 `=` 的区别是什么？**
     - `==` (两个等号) 是**关系运算符**，用来**判断**左右两边是否相等。它返回一个 `bool` 值。例如 `if (a == 5)` 是在问“a 等于 5 吗？”。
     - `=` (一个等号) 是**赋值运算符**，用来把右边的值**赋给**左边的变量。例如 `a = 5;` 是在命令“让 a 的值变成 5”。
     - **最常犯的错误**就是在需要判断的地方写了赋值，例如 `a = 5`，这会导致逻辑错误且很难发现！**请务必牢记！**
  2. `y++` **所在的这行代码结束后，** `y` **的值到底变了没有？**
     - **绝对变了！** 后置运算符的“后”指的是在“使用”之后，但在整个语句结束之前，变量的值就已经更新了。只要记住口诀“先用旧值，再变新值”就不会错。
  3. **任何数字都能转成** `bool` **类型吗？**
     - 是的。C++ 有一个规则：**非零即为真。**
     - 当一个整数或浮点数转换为 `bool` 类型时，只要它的值**不是** `0`，就转为 `true`。只有 `0` 会转为 `false`。
     - 例如：`(bool) -3` 的结果是 `true`，`(bool) 100` 的结果是 `true`，而 `(bool) 0` 的结果是 `false`。

---

6. 总结与练习

- **要点回顾**
  1. **算术运算符**：`+`, `-`, `*`, `/` (整除), `%` (取余)。
  2. **自增/自减**：`++` 和 `--` 是 `+1` 和 `-1` 的简写。
  3. **前置 vs 后置**：`++a` (先变后用)，`a++` (先用后变)。
  4. `bool` **类型**：专门存储 `true` (1) 和 `false` (0) 的类型，是判断的基础。
  5. **关系运算符**：`==`, `!=`, `>`, `<`, `>=`, `<=`，它们的结果都是 `bool` 类型。
  6. **优先级**：先算术，后关系。括号优先级最高。
- **拓展思考**
  我们已经学会了如何进行一次比较，比如 `a > b`。但如果我想判断“一个数 `x` 是否既大于 0 又小于 10”呢？这需要**两个**关系运算同时成立。要连接多个关系表达式，我们就需要学习下一讲的内容——**逻辑运算符** (`&&`, `||`, `!`)。它们是构建更复杂判断逻辑的桥梁。