C++ `if` 条件分支实战演练 讲义

一、 引言

问题引入：条件判断问题

在编写程序时，经常会出现根据条件不同，所做的事情也不同的情况，例如：

- **如果** 天在下雨，你**就**会带上雨伞
- **否则**（也就是天没下雨），你**就**不带伞。

这个“如果...就...否则...”的过程，就是我们无时无刻不在进行的**决策**。我们会根据不同的**条件**（是否下雨），选择执行不同的**动作**（带伞或不带伞）。

程序可以通过 `if` 条件分支结构 来拥有这种在分岔点做选择的能力，使我们能够：

- **判断用户输入**：判断用户输入的数字是否在某个范围内、输入的是奇数还是偶数。
- **做出决策**：根据不同的条件，执行完全不同的代码路径。
- **响应变化**：例如根据游戏角色的血量，决定是继续攻击还是播放阵亡动画。

二、 核心思想：如何解决条件判断问题

对于一个需要条件判断的问题，我们有如下的解题模板（**这是本讲义重要的部分，请务必理解！**）：

1. **【第一步：找到分岔点】**
   - 仔细阅读题目，找出决定程序走向的**所有关键条件**。这些就是程序需要做出选择的分岔点。
   - 例如：“判断奇偶性”的分岔点是“能否被2整除”；“判断成年人”的分岔点是“年龄是否大于等于18”。
2. **【第二步：写出判断分岔点的条件语句】**
   - 将这些用自然语言描述的分岔点，翻译成 C++ **条件表达式**。
   - 例如：需要判断分数在70到80之间 -> `score >= 70 && score <= 80`。
3. **【第三步：根据分岔点的数量和关系选择合适的`if`分支语句】**
   - **二选一**（是/否）：用 `if-else` 结构。（例如：判断一个数是 奇数/偶数）
   - **只关心一种情况**：只用一个 `if` 结构。（例如：只有及格了才发奖状，不及格什么都不做）
   - **多选关系**：根据题意使用多个`if` 语句进行处理。（例如：学生考试评级为A、B、C 时分别要做不同的事情，那么就需要三个 `if` 语句，`if(score == 'A'){}  if(score == 'B'){};  if(score == 'C'){};` 来分别处理每一种情况，其中 `score` 是储存考试评级的 `char` 变量）
4. **【第四步：实现各分支的逻辑代码】**
   - 在搭建好 `if`、`else if` 与 `else` 的结构框架后，需要在其对应的 `{}` 代码块中，**实现**当该分支条件被满足时，程序所需执行的**具体逻辑**。包括但不限于：执行数学运算、对变量进行赋值，或者调用 `cout` 将计算结果等输出。

三、 实战演练

接下来，我们列出几个上课中所讲解的案例，来深化`if`语句的使用。

案例一：奇偶判断 (二选一)

1. **分岔点**：一个数是“偶数”还是“奇数”。本质条件是“能否被2整除”。
2. **写出判断分岔的条件语句**：`num % 2 == 0`。如果为`true`，说明是偶数。
3. **分岔结构**：这是一个“是/否”二选一问题，使用`if-else`。
4. **程序代码**：

```
int main() {
    int num;
    cin >> num;
    if (num % 2 == 0) { // 如果“写出判断分岔的条件语句”为真
        cout << "偶数";  // 走“偶数”这条路
    } else {
        cout << "奇数";  // 否则，走“奇数”这条路
    }
    return 0;
}
```

案例二：景点门票

1. **分岔点**：票价是“半价”还是“全价”。条件是“年龄小于等于6” **或者** “年龄大于等于60”。
2. **写出判断分岔的条件语句**：这里的连接词是“或者”，翻译成 C++ 就是`||`。写出条件语句为：`age <= 6 || age >= 60`。
3. **分岔结构**：二选一问题，使用`if-else`。
4. **程序代码**：

```
if (age <= 6 || age >= 60) {
    cout << 25; // 半价
} else {
    cout << 50; // 全价
}
```

案例三：计算器 (多分支判断)

题目描述：设计一个计算器。让用户输入**num1+num2**、**num1-num2**、**num1*num2**、**num1/num2**形式的计算式。

我们的计算器便能够自动识别此时是要进行加法运算、减法运算、乘法运算，还是除法运算。最终输出计算结果。

1. **分岔点**：用户想进行哪种运算？取决于输入的运算符是`+`、`-`、`*`还是`/`。这是四个不同的岔路。
2. **写出判断分岔的条件语句**：`ch == '+'`、`ch == '-'`、`ch == '*'`、`ch == '/'`。
3. **分岔结构**：这里有多个独立的条件，每个条件都要做不同的事情，我们可以用一连串的`if`语句来分别检查。
4. **程序代码**：

```
int num1, num2;
char ch;
cin >> num1 >> ch >> num2;

if (ch == '+') {
    cout << num1 + num2;
}
if (ch == '-') {
    cout << num1 - num2;
}
if (ch == '*') {
    cout << num1 * num2;
}
if (ch == '/') {
    cout << (double)num1 / num2;
}
```

案例四：闰年判断

题目描述：请输入一个年份，判断该年是不是闰年。是的话输入该年份，不是的话输出"No"。其中能被4整除 **并且** 不能被100整除 **或者** 能被400整除 的是闰年。(这个判断条件稍显复杂，但是只要能写出判断分岔的条件语句，根据前面的条件判断问题解题模板，也能轻松解决。)

1. **分岔点**：这一年是“闰年”还是“平年”。
2. **写出判断分岔的所有条件语句**
   - 规则：(能被4整除 **并且** 不能被100整除) **或者** (能被400整除)。
   - 条件A（能被4整除）：`year % 4 == 0`
   - 条件B（不能被100整除）：`year % 100 != 0`
   - 条件C（能被400整除）：`year % 400 == 0`
   - 将条件A, B, C组合起来，注意括号代表优先级：`(A && B) || C`
   - 最终写出判断分岔的条件语句：`((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)`
3. **分岔结构**：二选一，用 `if-else`。
4. **程序代码**：

```
int year;
cin >> year;
if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)) {
    cout << year; // 是闰年，输出年份
} else {
    cout << "No"; // 不是闰年
}
```

案例五：鸡兔同笼

题目描述：已知鸡和兔的总数量为n，总腿数为 m。输入 n 和 m ,依次输出鸡的数目和兔的数目。如果无解，则输出No answer。

本问题将编程与数学的结合起来了。我们需要根据程序计算出来的结果去**验证计算结果的合理性**。我们可以列出两个方程：

$$鸡数量 + 兔数量 = n \ ①$$

$$2 * 鸡数量 + 4 * 兔数量 = m \ ②$$

1. **分岔点**：根据给定的头和腿的数量，算出来的鸡兔数量来判断是“有解”还是“无解”。
2. **写出判断分岔的条件语句**：列出方程后，进行变换后可以得到：`鸡 = (4*n-m)/2` 和 `兔 = n-鸡` 中，于是我们能推断出**无解**的条件：
   - `4*n-m` 必须是偶数，否则鸡的数量是小数。 -> `(4*n-m) % 2 != 0`
   - 算出来的鸡的数量不能是负数。 -> `鸡 < 0`
   - 算出来的兔的数量不能是负数。 -> `兔 < 0`
   - 这三个**无解**的条件，**只要满足任意一个**，就说明无解。所以用 `||` 连接。
   - 最终写出判断分岔的条件语句：`( (4*n-m) % 2 != 0 || 鸡 < 0 || 兔 < 0 )`
3. **分岔结构**：二选一，`if-else`。
4. **程序代码**：

```
int n, m, chicken, rabbit;
cin >> n >> m;

// 先用数学公式计算，对应方程 鸡 = (4*n-m)/2 和 兔 = n-鸡
chicken = (4 * n - m) / 2;
rabbit = n - chicken;

// 再用if进行合理性验证
if ((4 * n - m) % 2 != 0 || chicken < 0 || rabbit < 0) {
    cout << "No answer";
} else {
    cout << chicken << " " << rabbit;
}
```

四、 总结

- **掌握四步解题模板**：
  遇到条件判断问题。请遵循“**识别分岔点 -> 写出分岔点的条件表达式 -> 选择分支结构 -> 通过代码实现分支逻辑**”的流程，能确保你的思路清晰。
- **翻译条件表达式**：
  将自然语言条件翻译为代码的条件表达式要求熟练掌握，需要通过持续练习来巩固。
- **多使用括号**：
  在构建复杂的逻辑表达式时，使用 `()` 来指定运算优先级。能确保逻辑正确，也能让你的代码更加易于阅读。