C++ 分支语句

目录

---

1. 分支语句的概念

作用：根据条件决定程序执行路径
核心思想：通过布尔表达式（`true`/`false`）控制代码流程
应用场景：用户输入验证、游戏逻辑、算法选择等

---

2. `if` 语句

语法

```
if (condition) {
    // 条件为真时执行的代码
}
```

示例

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

int main() {
    int age = 18;

    if (age >= 18) {
        cout << "您已成年" << endl;
    }

    return 0;
}
```

关键点

- 当条件为真（非零）时执行代码块

- 如果代码块只有一行，可以省略大括号（但建议保留以提高可读性）

---

3. `if-else` 语句

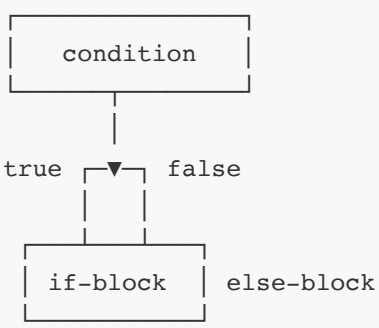
语法

```
if (condition) {
    // 条件为真时执行
} else {
    // 条件为假时执行
}
```

示例

```
int score = 85;
if (score >= 60) {
    cout << "及格" << endl;
} else {
    cout << "不及格" << endl;
}
```

流程图



---

4. `else if` 结构

语法

```
if (condition1) {
    // 条件1为真
} else if (condition2) {
    // 条件2为真
} else {
    // 所有条件都不满足
}
```

示例（成绩等级判断）

```cpp
int score = 85;

if (score >= 90) {
    cout << "A" << endl;
} else if (score >= 80) {
    cout << "B" << endl;  // 此分支将被执行
} else if (score >= 70) {
    cout << "C" << endl;
} else {
    cout << "D" << endl;
}
```

注意事项

- 按顺序检查条件，遇到第一个为真的条件后执行对应代码块
- 使用合理的条件顺序（建议从最严格的条件开始）

---

5. `switch` 语句

语法

```cpp
switch (expression) {
    case value1:
        // 代码
        break;
    case value2:
        // 代码
        break;
    default:
        // 默认代码
}
```

示例（星期判断）

```cpp
int day = 3;

switch (day) {
    case 1:
        cout << "Monday" << endl;
        break;
    case 2:
        cout << "Tuesday" << endl;
        break;
    case 3:
        cout << "Wednesday" << endl;  // 输出此项
```

```
        break;
    default:
        cout << "Invalid day" << endl;
}
```

重要特性

- `expression` 必须是整型或枚举类型
- `break` 用于退出switch结构（无break会导致"case穿透"）
- `default` 处理未匹配的情况

特殊用法（case穿透）

```
char grade = 'B';
switch (grade) {
    case 'A':
    case 'B':
    case 'C':
        cout << "合格";  // A/B/C都会执行此代码
        break;
    case 'D':
        cout << "不合格";
        break;
}
```

---

6. 三元条件运算符

语法

```
variable = (condition) ? true_expression : false_expression;
```

示例

```
int a = 10, b = 20;
int max = (a > b) ? a : b;  // 结果：20

cout << "绝对值：" << ((num < 0) ? -num : num);
```

适用场景

- 简单的条件赋值
- 需要表达式（而非语句）的场合

---

7. 总结与最佳实践

选择结构对比

结构	适用场景	优点
`if`	简单条件判断	灵活直观
`if-else if`	多条件分级判断	结构清晰
`switch`	精确值匹配（整型/枚举）	执行效率高
三元运算符	简单条件赋值	代码简洁

最佳实践

1. 优先使用最清晰的结构
2. 避免深层嵌套（超过3层建议重构）
3. 总是添加`default`处理未预见情况
4. 对浮点数比较使用范围检查而非`==`
5. 使用大括号明确代码块范围

常见错误

```
// 错误1：赋值运算符代替比较运算符
if (x = 5) {}   // 应改为 x == 5

// 错误2：忘记break导致case穿透
switch (val) {
    case 1: cout << "1";  // 缺少break会继续执行case2
    case 2: cout << "2";
}
```

练习建议

1. 编写程序判断闰年
2. 实现计算器基本运算选择
3. 制作简单的菜单选择系统
4. 编写成绩转换程序（百分制→字母等级）

通过这份讲稿，学习者可以系统地掌握C++中的各种分支结构，并理解如何根据实际需求选择最合适的条件判断方式。建议配合实际编码练习加深理解。