C++: Conditional (Ternary) Operator ? : — Complete Guide

1. Basic usage

The conditional operator returns one of two expressions based on a boolean condition.

```
int x = (a > b) ? a : b;
std::string s = ok ? "yes" : "no";
```

2. Readability and limits

Use for simple choices; avoid nesting which reduces clarity.

```
int sign = (x>0) ? 1 : (x<0 ? -1 : 0);  // nested ternary — can be confusing
```

3. Lvalue conditional (assignable result)

Conditional operator yields an lvalue when both second and third operands are lvalues of the same type.

```
(cond ? a : b) = value; // valid if a and b are lvalues of same type
```

4. Type and conversion rules

Second and third operands must be convertible to a common type. Usual arithmetic conversions apply for numeric types.

```
auto v = condition ? 1 : 2.0; // v is double (promoted)
```

5. Side effects and evaluation order

Only the selected expression is evaluated. Do not rely on both sides executing.

```
int f(){ std::cout<<"f\n"; return 1; }
int g(){ std::cout<<"g\n"; return 2; }
int x = cond ? f() : g(); // only f() or g() runs
```

6. Using ternary for initialization and return

Ternary is handy for compact initialization or returning values quickly.

```
return (x%2==0) ? "even" : "odd";
```

7. Best practices

- Prefer ternary for short, simple expressions. - Avoid side-effects in operands to keep behavior clear. - When nested, prefer named helper or if-else for clarity.