Boolean logical operators - AND, OR, NOT, XOR

Article • 12/02/2022 • 7 minutes to read

The logical Boolean operators perform logical operations with bool operands. The operators include the unary logical negation (`!`), binary logical AND (`&`), OR (`|`), and exclusive OR (`^`), and the binary conditional logical AND (`&&`) and OR (`||`).

- Unary ! (logical negation) operator.
- Binary & (logical AND), | (logical OR), and ^ (logical exclusive OR) operators. Those operators always evaluate both operands.
- Binary && (conditional logical AND) and || (conditional logical OR) operators. Those operators evaluate the right-hand operand only if it's necessary.

For operands of the integral numeric types, the `&`, `|`, and `^` operators perform bitwise logical operations. For more information, see Bitwise and shift operators.

Logical negation operator !

The unary prefix `!` operator computes logical negation of its operand. That is, it produces `true`, if the operand evaluates to `false`, and `false`, if the operand evaluates to `true`:

C#

```csharp
bool passed = false;
Console.WriteLine(!passed);  // output: True
Console.WriteLine(!true);    // output: False
```

The unary postfix `!` operator is the null-forgiving operator.

Logical AND operator &

The `&` operator computes the logical AND of its operands. The result of `x & y` is `true` if both `x` and `y` evaluate to `true`. Otherwise, the result is `false`.

The `&` operator evaluates both operands even if the left-hand operand evaluates to `false`, so that the operation result is `false` regardless of the value of the right-hand operand.

In the following example, the right-hand operand of the `&` operator is a method call, which is performed regardless of the value of the left-hand operand:

```C#
bool SecondOperand()
{
    Console.WriteLine("Second operand is evaluated.");
    return true;
}

bool a = false & SecondOperand();
Console.WriteLine(a);
// Output:
// Second operand is evaluated.
// False

bool b = true & SecondOperand();
Console.WriteLine(b);
// Output:
// Second operand is evaluated.
// True
```

The conditional logical AND operator `&&` also computes the logical AND of its operands, but doesn't evaluate the right-hand operand if the left-hand operand evaluates to `false`.

For operands of the integral numeric types, the `&` operator computes the bitwise logical AND of its operands. The unary `&` operator is the address-of operator.

Logical exclusive OR operator ^

The `^` operator computes the logical exclusive OR, also known as the logical XOR, of its operands. The result of `x ^ y` is `true` if `x` evaluates to `true` and `y` evaluates to `false`, or `x` evaluates to `false` and `y` evaluates to `true`. Otherwise, the result is `false`. That is, for the `bool` operands, the `^` operator computes the same result as the inequality operator `!=`.

```C#
Console.WriteLine(true ^ true);    // output: False
Console.WriteLine(true ^ false);   // output: True
Console.WriteLine(false ^ true);   // output: True
Console.WriteLine(false ^ false);  // output: False
```

For operands of the integral numeric types, the `^` operator computes the bitwise logical exclusive OR of its operands.

Logical OR operator |

The `|` operator computes the logical OR of its operands. The result of `x | y` is `true` if either `x` or `y` evaluates to `true`. Otherwise, the result is `false`.

The `|` operator evaluates both operands even if the left-hand operand evaluates to `true`, so that the operation result is `true` regardless of the value of the right-hand operand.

In the following example, the right-hand operand of the `|` operator is a method call, which is performed regardless of the value of the left-hand operand:

C#

```
bool SecondOperand()
{
    Console.WriteLine("Second operand is evaluated.");
    return true;
}

bool a = true | SecondOperand();
Console.WriteLine(a);
// Output:
// Second operand is evaluated.
// True

bool b = false | SecondOperand();
Console.WriteLine(b);
// Output:
// Second operand is evaluated.
// True
```

The conditional logical OR operator `||` also computes the logical OR of its operands, but doesn't evaluate the right-hand operand if the left-hand operand evaluates to `true`.

For operands of the integral numeric types, the `|` operator computes the bitwise logical OR of its operands.

Conditional logical AND operator &&

The conditional logical AND operator `&&`, also known as the "short-circuiting" logical AND operator, computes the logical AND of its operands. The result of `x && y` is `true` if both `x` and `y` evaluate to `true`. Otherwise, the result is `false`. If `x` evaluates to `false`, `y` isn't evaluated.

In the following example, the right-hand operand of the `&&` operator is a method call, which isn't performed if the left-hand operand evaluates to `false`:

C#

```
bool SecondOperand()
{
    Console.WriteLine("Second operand is evaluated.");
    return true;
}

bool a = false && SecondOperand();
Console.WriteLine(a);
// Output:
// False

bool b = true && SecondOperand();
Console.WriteLine(b);
// Output:
// Second operand is evaluated.
// True
```

The logical AND operator `&` also computes the logical AND of its operands, but always evaluates both operands.

Conditional logical OR operator ||

The conditional logical OR operator `||`, also known as the "short-circuiting" logical OR operator, computes the logical OR of its operands. The result of `x || y` is `true` if either `x` or `y` evaluates to `true`. Otherwise, the result is `false`. If `x` evaluates to `true`, `y` isn't evaluated.

In the following example, the right-hand operand of the `||` operator is a method call, which isn't performed if the left-hand operand evaluates to `true`:

C#

```
bool SecondOperand()
{
    Console.WriteLine("Second operand is evaluated.");
    return true;
}

bool a = true || SecondOperand();
Console.WriteLine(a);
// Output:
// True

bool b = false || SecondOperand();
Console.WriteLine(b);
// Output:
// Second operand is evaluated.
// True
```

The logical OR operator `|` also computes the logical OR of its operands, but always evaluates both operands.

Nullable Boolean logical operators

For `bool?` operands, the & (logical AND) and | (logical OR) operators support the three-valued logic as follows:

- The `&` operator produces `true` only if both its operands evaluate to `true`. If either `x` or `y` evaluates to `false`, `x & y` produces `false` (even if another operand evaluates to `null`). Otherwise, the result of `x & y` is `null`.
- The `|` operator produces `false` only if both its operands evaluate to `false`. If either `x` or `y` evaluates to `true`, `x | y` produces `true` (even if another operand evaluates to `null`). Otherwise, the result of `x | y` is `null`.

The following table presents that semantics:

x	**y**	**x&y**	**x|y**
true	true	true	true
true	false	false	true
true	null	null	true

x	y	x&y	x|y
false	true	false	true
false	false	false	false
false	null	false	null
null	true	null	true
null	false	false	null
null	null	null	null

The behavior of those operators differs from the typical operator behavior with nullable value types. Typically, an operator that is defined for operands of a value type can be also used with operands of the corresponding nullable value type. Such an operator produces `null` if any of its operands evaluates to `null`. However, the `&` and `|` operators can produce non-null even if one of the operands evaluates to `null`. For more information about the operator behavior with nullable value types, see the Lifted operators section of the Nullable value types article.

You can also use the `!` and `^` operators with `bool?` operands, as the following example shows:

C#

```
bool? test = null;
Display(!test);         // output: null
Display(test ^ false);  // output: null
Display(test ^ null);   // output: null
Display(true ^ null);   // output: null

void Display(bool? b) => Console.WriteLine(b is null ? "null" :
b.Value.ToString());
```

The conditional logical operators `&&` and `||` don't support `bool?` operands.

Compound assignment

For a binary operator `op`, a compound assignment expression of the form

C#

```
x op= y
```

is equivalent to

C#

```
x = x op y
```

except that `x` is only evaluated once.

The `&`, `|`, and `^` operators support compound assignment, as the following example shows:

C#

```
bool test = true;
test &= false;
Console.WriteLine(test);  // output: False

test |= true;
Console.WriteLine(test);  // output: True

test ^= false;
Console.WriteLine(test);  // output: True
```

Note

The conditional logical operators `&&` and `||` don't support compound assignment.

Operator precedence

The following list orders logical operators starting from the highest precedence to the lowest:

- Logical negation operator `!`
- Logical AND operator `&`
- Logical exclusive OR operator `^`
- Logical OR operator `|`
- Conditional logical AND operator `&&`

- Conditional logical OR operator `||`

Use parentheses, `()`, to change the order of evaluation imposed by operator precedence:

```csharp
Console.WriteLine(true | true & false);   // output: True
Console.WriteLine((true | true) & false); // output: False

bool Operand(string name, bool value)
{
    Console.WriteLine($"Operand {name} is evaluated.");
    return value;
}

var byDefaultPrecedence = Operand("A", true) || Operand("B", true) &&
Operand("C", false);
Console.WriteLine(byDefaultPrecedence);
// Output:
// Operand A is evaluated.
// True

var changedOrder = (Operand("A", true) || Operand("B", true)) && Operand("C",
false);
Console.WriteLine(changedOrder);
// Output:
// Operand A is evaluated.
// Operand C is evaluated.
// False
```

For the complete list of C# operators ordered by precedence level, see the Operator precedence section of the C# operators article.

Operator overloadability

A user-defined type can overload the `!`, `&`, `|`, and `^` operators. When a binary operator is overloaded, the corresponding compound assignment operator is also implicitly overloaded. A user-defined type can't explicitly overload a compound assignment operator.

A user-defined type can't overload the conditional logical operators `&&` and `||`. However, if a user-defined type overloads the true and false operators and the `&` or `|` operator in a certain way, the `&&` or `||` operation, respectively, can be evaluated for the operands of that type. For more information, see the User-defined conditional logical operators section of the C# language specification.

C# language specification

For more information, see the following sections of the C# language specification:

- Logical negation operator
- Logical operators
- Conditional logical operators
- Compound assignment

See also

- C# reference
- C# operators and expressions
- Bitwise and shift operators