**C# operators and expressions (C# reference)**

* 5 minutes to read

C# provides a number of operators. Many of them are supported by the [built-in types](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types) and allow you to perform basic operations with values of those types. Those operators include the following groups:

* [Arithmetic operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators) that perform arithmetic operations with numeric operands
* [Comparison operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators) that compare numeric operands
* [Boolean logical operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators) that perform logical operations with [bool](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/bool) operands
* [Bitwise and shift operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators) that perform bitwise or shift operations with operands of the integral types
* [Equality operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators) that check if their operands are equal or not

Typically, you can [overload](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading) those operators, that is, specify the operator behavior for the operands of a user-defined type.

The simplest C# expressions are literals (for example, [integer](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types#integer-literals) and [real](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types#real-literals) numbers) and names of variables. You can combine them into complex expressions by using operators. Operator [precedence](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/#operator-precedence) and [associativity](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/#operator-associativity) determine the order in which the operations in an expression are performed. You can use parentheses to change the order of evaluation imposed by operator precedence and associativity.

In the following code, examples of expressions are at the right-hand side of assignments:

C#Copy

int a, b, c;

a = 7;

b = a;

c = b++;

b = a + b \* c;

c = a >= 100 ? b : c / 10;

a = (int)Math.Sqrt(b \* b + c \* c);

string s = "String literal";

char l = s[s.Length - 1];

var numbers = new List<int>(new[] { 1, 2, 3 });

b = numbers.FindLast(n => n > 1);

Typically, an expression produces a result and can be included in another expression. A [void](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/void) method call is an example of an expression that doesn't produce a result. It can be used only as a [statement](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/statements), as the following example shows:

C#Copy

Console.WriteLine("Hello, world!");

Here are some other kinds of expressions that C# provides:

* [Interpolated string expressions](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated) that provide convenient syntax to create formatted strings:

C#Copy

Run

var r = 2.3;

var message = $"The area of a circle with radius {r} is {Math.PI \* r \* r:F3}.";

Console.WriteLine(message);

// Output:

// The area of a circle with radius 2.3 is 16.619.

* [Lambda expressions](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-expressions) that allow you to create anonymous functions:

C#Copy

Run

int[] numbers = { 2, 3, 4, 5 };

var maximumSquare = numbers.Max(x => x \* x);

Console.WriteLine(maximumSquare);

// Output:

// 25

* [Query expressions](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/query-keywords) that allow you to use query capabilities directly in C#:

C#Copy

Run

var scores = new[] { 90, 97, 78, 68, 85 };

IEnumerable<int> highScoresQuery =

from score in scores

where score > 80

orderby score descending

select score;

Console.WriteLine(string.Join(" ", highScoresQuery));

// Output:

// 97 90 85

You can use an [expression body definition](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/expression-bodied-members) to provide a concise definition for a method, constructor, property, indexer, or finalizer.

**Operator precedence**

In an expression with multiple operators, the operators with higher precedence are evaluated before the operators with lower precedence. In the following example, the multiplication is performed first because it has higher precedence than addition:

C#Copy

Run

var a = 2 + 2 \* 2;

Console.WriteLine(a); // output: 6

Use parentheses to change the order of evaluation imposed by operator precedence:

C#Copy

Run

var a = (2 + 2) \* 2;

Console.WriteLine(a); // output: 8

The following table lists the C# operators starting with the highest precedence to the lowest. The operators within each row have the same precedence.

| **TABLE 1** | |
| --- | --- |
| **Operators** | **Category or name** |
| [x.y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#member-access-expression-), [f(x)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#invocation-expression-), [a[i]](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#indexer-operator-), [x?.y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators" \l "null-conditional-operators--and-), [x?[y]](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#null-conditional-operators--and-), [x++](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#increment-operator-), [x--](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#decrement-operator---), [x!](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-forgiving), [new](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/new-operator), [typeof](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#typeof-operator), [checked](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/checked), [unchecked](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/unchecked), [default](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/default), [nameof](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/nameof), [delegate](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/delegate-operator), [sizeof](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/sizeof), [stackalloc](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/stackalloc), [x->y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/pointer-related-operators#pointer-member-access-operator--) | Primary |
| [+x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#unary-plus-and-minus-operators), [-x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#unary-plus-and-minus-operators), [!x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-negation-operator-), [~x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#bitwise-complement-operator-), [++x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#increment-operator-), [--x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#decrement-operator---), [^x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#index-from-end-operator-), [(T)x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#cast-expression), [await](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await), [&x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/pointer-related-operators#address-of-operator-), [\*x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/pointer-related-operators#pointer-indirection-operator-), [true and false](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/true-false-operators) | Unary |
| [x..y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#range-operator-) | Range |
| [switch](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/switch-expression) | switch expression |
| [with](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/with-expression) | with expression |
| [x \* y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#multiplication-operator-), [x / y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#division-operator-), [x % y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#remainder-operator-) | Multiplicative |
| [x + y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#addition-operator-), [x – y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#subtraction-operator--) | Additive |
| [x << y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#left-shift-operator-), [x >> y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#right-shift-operator-) | Shift |
| [x < y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#less-than-operator-), [x > y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#greater-than-operator-), [x <= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#less-than-or-equal-operator-), [x >= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#greater-than-or-equal-operator-), [is](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#is-operator), [as](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#as-operator) | Relational and type-testing |
| [x == y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators#equality-operator-), [x != y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators#inequality-operator-) | Equality |
| x & y | [Boolean logical AND](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-and-operator-) or [bitwise logical AND](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#logical-and-operator-) |
| x ^ y | [Boolean logical XOR](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-exclusive-or-operator-) or [bitwise logical XOR](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#logical-exclusive-or-operator-) |
| x | y | [Boolean logical OR](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-or-operator-) or [bitwise logical OR](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#logical-or-operator-) |
| [x && y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#conditional-logical-and-operator-) | Conditional AND |
| [x || y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#conditional-logical-or-operator-) | Conditional OR |
| [x ?? y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator) | Null-coalescing operator |
| [c ? t : f](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator) | Conditional operator |
| [x = y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/assignment-operator), [x += y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [x -= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [x \*= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [x /= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [x %= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [x &= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#compound-assignment), [x |= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#compound-assignment), [x ^= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#compound-assignment), [x <<= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#compound-assignment), [x >>= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#compound-assignment), [x ??= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator), [=>](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-operator) | Assignment and lambda declaration |

**Operator associativity**

When operators have the same precedence, associativity of the operators determines the order in which the operations are performed:

* *Left-associative* operators are evaluated in order from left to right. Except for the [assignment operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/assignment-operator) and the [null-coalescing operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator), all binary operators are left-associative. For example, a + b - c is evaluated as (a + b) - c.
* *Right-associative* operators are evaluated in order from right to left. The assignment operators, the null-coalescing operators, and the [conditional operator ?:](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator) are right-associative. For example, x = y = z is evaluated as x = (y = z).

Use parentheses to change the order of evaluation imposed by operator associativity:

C#Copy

Run

int a = 13 / 5 / 2;

int b = 13 / (5 / 2);

Console.WriteLine($"a = {a}, b = {b}"); // output: a = 1, b = 6

**Operand evaluation**

Unrelated to operator precedence and associativity, operands in an expression are evaluated from left to right. The following examples demonstrate the order in which operators and operands are evaluated:

| **OPERAND EVALUATION** | |
| --- | --- |
| **Expression** | **Order of evaluation** |
| a + b | a, b, + |
| a + b \* c | a, b, c, \*, + |
| a / b + c \* d | a, b, /, c, d, \*, + |
| a / (b + c) \* d | a, b, c, +, /, d, \* |

Typically, all operator operands are evaluated. However, some operators evaluate operands conditionally. That is, the value of the leftmost operand of such an operator defines if (or which) other operands should be evaluated. These operators are the conditional logical [AND (&&)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#conditional-logical-and-operator-) and [OR (||)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#conditional-logical-or-operator-) operators, the [null-coalescing operators ?? and ??=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator), the [null-conditional operators ?. and ?[]](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#null-conditional-operators--and-), and the [conditional operator ?:](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator). For more information, see the description of each operator.

**C# language specification**

For more information, see the following sections of the [C# language specification](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction):

* [Expressions](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/expressions)
* [Operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/expressions#operators)

**See also**

* [C# reference](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/)
* [Operator overloading](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading)
* [Expression trees](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/expression-trees/)