Operator overloading - predefined unary, arithmetic, equality and comparison operators

Article • 12/02/2022 • 3 minutes to read

A user-defined type can overload a predefined C# operator. That is, a type can provide the custom implementation of an operation in case one or both of the operands are of that type. The Overloadable operators section shows which C# operators can be overloaded.

Use the `operator` keyword to declare an operator. An operator declaration must satisfy the following rules:

- It includes both a `public` and a `static` modifier.
- A unary operator has one input parameter. A binary operator has two input parameters. In each case, at least one parameter must have type `T` or `T?` where `T` is the type that contains the operator declaration.

The following example defines a simplified structure to represent a rational number. The structure overloads some of the arithmetic operators:

C#

```csharp
public readonly struct Fraction
{
    private readonly int num;
    private readonly int den;

    public Fraction(int numerator, int denominator)
    {
        if (denominator == 0)
        {
            throw new ArgumentException("Denominator cannot be zero.",
nameof(denominator));
        }
        num = numerator;
        den = denominator;
    }

    public static Fraction operator +(Fraction a) => a;
    public static Fraction operator -(Fraction a) => new Fraction(-a.num,
a.den);

    public static Fraction operator +(Fraction a, Fraction b)
        => new Fraction(a.num * b.den + b.num * a.den, a.den * b.den);

    public static Fraction operator -(Fraction a, Fraction b)
        => a + (-b);

    public static Fraction operator *(Fraction a, Fraction b)
        => new Fraction(a.num * b.num, a.den * b.den);

    public static Fraction operator /(Fraction a, Fraction b)
    {
        if (b.num == 0)
        {
            throw new DivideByZeroException();
        }
        return new Fraction(a.num * b.den, a.den * b.num);
    }

    public override string ToString() => $"{num} / {den}";
}

public static class OperatorOverloading
{
    public static void Main()
    {
        var a = new Fraction(5, 4);
        var b = new Fraction(1, 2);
        Console.WriteLine(-a);   // output: -5 / 4
        Console.WriteLine(a + b);  // output: 14 / 8
        Console.WriteLine(a - b);  // output: 6 / 8
        Console.WriteLine(a * b);  // output: 5 / 8
        Console.WriteLine(a / b);  // output: 10 / 4
    }
}
```

You could extend the preceding example by defining an implicit conversion from `int` to `Fraction`. Then, overloaded operators would support arguments of those two types. That is, it would become possible to add an integer to a fraction and obtain a fraction as a result.

You also use the `operator` keyword to define a custom type conversion. For more information, see User-defined conversion operators.

Overloadable operators

The following table shows the operators that can be overloaded:

Operators	Notes
+x, -x, !x, ~x, ++, --, true, false	The `true` and `false` operators must be overloaded together.
x + y, x - y, x * y, x / y, x % y, x & y, x | y, x ^ y, x << y, x >> y, x >>> y	
x == y, x != y, x < y, x > y, x <= y, x >= y	Must be overloaded in pairs as follows: `==` and `!=`, `<` and `>`, `<=` and `>=`.

Non overloadable operators

The following table shows the operators that can't be overloaded:

Operators	Alternatives
x && y, x || y	Overload both the true and false operators and the & or | operators. For more information, see User-defined conditional logical operators.
a[i], a?[i]	Define an indexer.
(T)x	Define custom type conversions that can be performed by a cast expression. For more information, see User-defined conversion operators.
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=	Overload the corresponding binary operator. For example, when you overload the binary `+` operator, `+=` is implicitly overloaded.
^x, x = y, x.y, x?.y, c ? t : f, x ?? y, ??= y, x..y, x->y, =>, f(x), as, await, checked, unchecked, default, delegate, is, nameof, new, sizeof, stackalloc, switch, typeof, with	None.

C# language specification

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

- Operator overloading
- Operators

See also

- C# reference
- C# operators and expressions
- User-defined conversion operators
- Design guidelines - Operator overloads
- Design guidelines - Equality operators
- Why are overloaded operators always static in C#?