Operators in C#

Arithmetic Operators

`+` : Addition

`-` : Subtraction

`*` : Multiplication

`/` : Division

`%` : Modulus (remainder after division)

Assignment Operators

`=` : Simple assignment

`+=` : Addition assignment

`-=` : Subtraction assignment

`*=` : Multiplication assignment

`/=` : Division assignment

`%=` : Modulus assignment

Comparison Operators

`==` : Equal to

`!=` : Not equal to

`>` : Greater than

`<` : Less than

`>=` : Greater than or equal to

`<=` : Less than or equal to

Logical Operators

`&&` : Logical AND

`||` : Logical OR

`!` : Logical NOT

Bitwise Operators

`&` : Bitwise AND

`|` : Bitwise OR

`^` : Bitwise XOR

`~` : Bitwise NOT

`<<` : Left shift

`>>` : Right shift

Unary Operators

`+` : Unary plus

`-` : Unary minus

`++` : Increment

`--` : Decrement

`!` : Logical NOT

Ternary Operator

`?:` : Conditional (ternary) operator

Null-coalescing Operators

`??` : Returns the left-hand operand if it is not null; otherwise, it returns the right-hand operand

`??=` : Assigns the right-hand operand to the left-hand operand if the left-hand operand is null

Type Operators

`is` : Checks if an object is compatible with a given type

`as` : Performs a type conversion

Other Miscellaneous Operators

`sizeof` : Returns the size of a type in bytes

`typeof` : Returns the `System.Type` object for a type

`&` : Address-of operator

`*` : Pointer dereference operator (unsafe code context)

Overloadable Operators

C# allows overloading of many operators, enabling them to be redefined for user-defined types:

Most arithmetic, comparison, and bitwise operators can be overloaded.

Logical operators `&&` and `||` cannot be directly overloaded, but can be indirectly overloaded by overloading `&` and `|`.

The assignment operator `=` cannot be overloaded, but compound assignment operators (`+=`, `-=`, etc.) can be overloaded.

The `[]` (indexer) operator can be overloaded to allow custom indexing for objects.

Example of Operator Overloading:

```csharp
public class Complex
{
   public double Real { get; set; }
   public double Imaginary { get; set; }

   public static Complex operator +(Complex c1, Complex c2)
   {
      return new Complex { Real = c1.Real + c2.Real, Imaginary = c1.Imaginary + c2.Imaginary };
   }
}
```

In this example, the `+` operator is overloaded to add two `Complex` objects.