Operators in C#

In C#, an **operator** is a symbol that performs operations on variables and values. Operators allow you to manipulate data and perform various computations. Operators are categorized based on the operations they perform. Here's a comprehensive overview of different types of operators available in C# with examples and explanations.

---

1. Arithmetic Operators

Arithmetic operators are used to perform mathematical operations.

Operators

- **Addition (+)**: Adds two operands.
- **Subtraction (-)**: Subtracts the second operand from the first.
- **Multiplication (*)**: Multiplies two operands.
- **Division (/)**: Divides the first operand by the second.
- **Modulus (%)**: Returns the remainder of a division operation.

Example

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;
        int b = 5;

        Console.WriteLine($"Addition: {a + b}");       // Outputs: Addition: 15
        Console.WriteLine($"Subtraction: {a - b}");   // Outputs: Subtraction: 5
        Console.WriteLine($"Multiplication: {a * b}");// Outputs: Multiplication: 50
        Console.WriteLine($"Division: {a / b}");      // Outputs: Division: 2
        Console.WriteLine($"Modulus: {a % b}");       // Outputs: Modulus: 0
    }}
```

2. Relational Operators

Relational operators are used to compare two values.

Operators

- **Equal to (==)**: Checks if two operands are equal.
- **Not equal to (!=)**: Checks if two operands are not equal.
- **Greater than (>)**: Checks if the first operand is greater than the second.
- **Less than (<)**: Checks if the first operand is less than the second.
- **Greater than or equal to (>=)**: Checks if the first operand is greater than or equal to the second.
- **Less than or equal to (<=)**: Checks if the first operand is less than or equal to the second.

Example

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;
        int b = 5;

        Console.WriteLine($"Equal to: {a == b}");            // Outputs: Equal to: False
        Console.WriteLine($"Not equal to: {a != b}");        // Outputs: Not equal to: True
        Console.WriteLine($"Greater than: {a > b}");         // Outputs: Greater than: True
        Console.WriteLine($"Less than: {a < b}");            // Outputs: Less than: False
        Console.WriteLine($"Greater than or equal to: {a >= b}"); // Outputs: Greater than or equal to: True
        Console.WriteLine($"Less than or equal to: {a <= b}");    // Outputs: Less than or equal to: False
    }
}
```

3. Logical Operators

Logical operators are used to combine or invert boolean values.

Operators

- **Logical AND (&&)**: Returns true if both operands are true.
- **Logical OR (||)**: Returns true if at least one operand is true.
- **Logical NOT (!)**: Inverts the boolean value of the operand.

Example

```
using System;
class Program
{
    static void Main(string[] args)
    {
        bool a = true;
        bool b = false;

        Console.WriteLine($"Logical AND: {a && b}"); // Outputs: Logical AND: False
        Console.WriteLine($"Logical OR: {a || b}");  // Outputs: Logical OR: True
        Console.WriteLine($"Logical NOT: {!a}");     // Outputs: Logical NOT: False
    }
}
```

4. Assignment Operators

Assignment operators are used to assign values to variables.

Operators

- **Assignment (=)**: Assigns a value to a variable.
- **Addition assignment (+=)**: Adds and assigns.
- **Subtraction assignment (-=)**: Subtracts and assigns.
- **Multiplication assignment (*=)**: Multiplies and assigns.
- **Division assignment (/=)**: Divides and assigns.
- **Modulus assignment (%=)**: Computes the modulus and assigns.

Example

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;

        a += 5;   // Equivalent to a = a + 5
        Console.WriteLine($"Addition Assignment: {a}"); // Outputs: Addition Assignment: 15

        a -= 3;   // Equivalent to a = a - 3
        Console.WriteLine($"Subtraction Assignment: {a}"); // Outputs: Subtraction Assignment: 12

        a *= 2;   // Equivalent to a = a * 2
        Console.WriteLine($"Multiplication Assignment: {a}"); // Outputs: Multiplication Assignment: 24

        a /= 4;   // Equivalent to a = a / 4
        Console.WriteLine($"Division Assignment: {a}"); // Outputs: Division Assignment: 6

        a %= 5;   // Equivalent to a = a % 5
        Console.WriteLine($"Modulus Assignment: {a}"); // Outputs: Modulus Assignment: 1
    }
}
```

5. Increment and Decrement Operators

These operators are used to increment or decrement a variable's value by 1.

Operators

- **Increment (++)**: Increases the value of a variable by 1.
- **Decrement (--)**: Decreases the value of a variable by 1.

Example

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;

        Console.WriteLine($"Initial value: {a}"); // Outputs: Initial value: 10

        a++;   // Increment
        Console.WriteLine($"After increment: {a}"); // Outputs: After increment: 11

        a--;   // Decrement
        Console.WriteLine($"After decrement: {a}"); // Outputs: After decrement: 10
    }
}
```

6. Conditional (Ternary) Operator

The conditional operator (`?:`) is a shorthand for an `if-else` statement.

Syntax

```
condition ? expressionIfTrue : expressionIfFalse;
```

Example

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int a = 10;
        string result = (a > 5) ? "Greater than 5" : "Not greater than 5";

        Console.WriteLine(result); // Outputs: Greater than 5
    }
}
```

7. Bitwise Operators

Bitwise operators perform operations on the binary representations of integers.

Operators

- **AND (&)**: Performs a bitwise AND.
- **OR (|)**: Performs a bitwise OR.
- **XOR (^)**: Performs a bitwise XOR.
- **Complement (~)**: Inverts all the bits.
- **Left Shift (<<)**: Shifts bits to the left.
- **Right Shift (>>)**: Shifts bits to the right.

Example

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 5;    // 00000101 in binary
        int b = 3;    // 00000011 in binary

        Console.WriteLine($"AND: {a & b}"); // Outputs: AND: 1 (00000001 in binary)
        Console.WriteLine($"OR: {a | b}");  // Outputs: OR: 7 (00000111 in binary)
        Console.WriteLine($"XOR: {a ^ b}"); // Outputs: XOR: 6 (00000110 in binary)
        Console.WriteLine($"Complement: {~a}"); // Outputs: Complement: -6 (inverts bits)
        Console.WriteLine($"Left Shift: {a << 1}"); // Outputs: Left Shift: 10 (00001010 in binary)
        Console.WriteLine($"Right Shift: {a >> 1}"); // Outputs: Right Shift: 2 (00000010 in binary)
    }
}
```

8. Null Coalescing Operators

These operators are used to provide default values for nullable types.

Operators

- **Null Coalescing (??)**: Returns the left-hand operand if it is not null; otherwise, returns the right-hand operand.
- **Null Coalescing Assignment (??=)**: Assigns the right-hand operand to the left-hand operand only if the left-hand operand is null.

Example

```
using System;
class Program
{
    static void Main(string[] args)
    {
        string name = null;

        // Null Coalescing
        string displayName = name ?? "Default Name";
        Console.WriteLine(displayName); // Outputs: Default Name

        // Null Coalescing Assignment
        name ??= "Assigned Name";
        Console.WriteLine(name); // Outputs: Assigned Name
    }
}
```

Summary

Operators in C# provide the fundamental building blocks for performing operations on data. They enable arithmetic, comparison, logical reasoning, and manipulation of data, allowing developers to write complex and efficient code. Understanding and using these operators effectively is crucial for writing robust C# applications.