# Operators

### Arithmetic Operators

All the operators work in the expected way and can be applied to any built-in numeric data type. The modulus operator works on integers as well as floats. Unlike VisualBasic, there is no exponent operator.

|  |  |
| --- | --- |
| Operator | Meaning |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| -- | Decrement |

### Relational/Comparison Operators

C# relational operators allow you to compare two similar types and return a bool.

|  |  |
| --- | --- |
| Operator | Description |
| == | Returns true if the two values are the same |
| != | Returns true if the two values are not the same |
| < | Returns true if one value is smaller than the other one |
| > | Returns true if one value is greater than the other one |
| <= | Returns true if one value is smaller or equal to the other one |
| >= | Returns true if one value is greater or equal to the other one |

### Bitwise Operators

C# bitwise operators allow you to manipulate values at the byte level. These specialized operators will not be used in this course.

|  |  |
| --- | --- |
| Operator | Description |
| & | Bitwise AND of two values |
| | | Bitwise OR of two values |
| ^ | Bitwise XOR of two values |
| ~ | Bitwise NOT of a values |
| << | Left-shift operator |
| >> | Right-shift operator |

### Sample code

byte a = 14;

byte b = 9;

Console.WriteLine(a & b); // and (00001110 & 00001001) = 00001000 = 8

Console.WriteLine(a | b); // or (00001110 | 00001001) = 00001111 = 15

Console.WriteLine(a ^ b); // xor (00001110 ^ 00001001) = 00000111 = 7

Console.WriteLine(~a); // negation of a

Console.WriteLine(a << 2); // multiplies a by 22 00001110 << 2 = 00111000 = 56

Console.WriteLine(a >> 2); // divides a by 22 remainder is discarded 00001110 >> 2 = 00000011 = 3

//output

// 8

// 15

// 7

// -15

// 56

// 3

### Conditional Operators

C# Conditional operators allow you to manipulate boolean values or expression. These are used to build more complex logic.

|  |  |
| --- | --- |
| Operator | Description |
| && | Returns true only if both operands are true |
| || | Return true if either operand is true |

### Operator Precedence (Page 705-706 in text)

The table below show the precedence of operators

|  |  |
| --- | --- |
| Category | Operator |
| Primary | x.y f(x) a[x] x++ x-- new typeof checked unchecked |
| Unary | + - ! ~ ++x --x (T)x |
| Multiplicative | \* / % |
| Additive | + - |
| Shift | << >> |
| Relational and Type checking | < > <= >= is as |
| Equality | == != |
| Logical AND | & |
| Logical XOR | ^ |
| Logical OR | | |
| Conditional AND | && |
| Conditional OR | || |
| Conditional | ?: |
| Assignment | = \*= /= %= += <<= >>= &= ^= |= |

All operators are left-associative except the assign and the conditional operator

e.g. a + b + c -> (a + b) + c [Left-associativity]

e.g. x = y = z -> x = (y = z) [Right-associativity]

Precedence and associativity can be controlled by parenthesis