**C# Operators**

C# provides a large set of operators, which are symbols that specify which operations to perform in an expression. Operations on integral types such as **==**, **!=**, **<**, **>**, **<=**, **>=**, **binary +**, **binary -**, **^**, **&**, **|**, **~**, **++**, **--**, and **sizeof()** are generally allowed on enumerations. In addition, many operators can be overloaded by the user, thus changing their meaning when applied to a user-defined type.

The following table lists the C# operators grouped in order of precedence. Operators within each group have equal precedence.

|  |  |
| --- | --- |
| **Operator category** | **Operators** |
| Primary | x.y  f(x)  a[x]  x++  x--  new  typeof  checked  unchecked  default(T)  delegate  -> |
| Unary | +  -  !  ~  ++x  --x  (T)x  true  false  &  sizeof |
| Multiplicative | \*  /  % |
| Additive | +  - |
| Shift | <<  >> |
| Relational and type testing | <  >  <=  >=  is  as |
| Equality | ==  != |
| Logical AND | & |
| Logical XOR | ^ |
| Logical OR | | |
| Conditional AND | && |
| Conditional OR | || |
| Null-coalescing | ?? |
| Conditional | ?: |
| Assignment and lambda expression | =  +=  -=  \*=  /=  %=  &=  |=  ^=  <<=  >>=  => Lambda Operator |

Arithmetic Overflow

The arithmetic operators ([+](https://msdn.microsoft.com/en-us/library/k1a63xkz(v=vs.100).aspx), [-](https://msdn.microsoft.com/en-us/library/wch5w409(v=vs.100).aspx), [\*](https://msdn.microsoft.com/en-us/library/z19tbbca(v=vs.100).aspx), [/](https://msdn.microsoft.com/en-us/library/3b1ff23f(v=vs.100).aspx)) can produce results that are outside the range of possible values for the numeric type involved. You should refer to the section on a particular operator for details, but in general:

* Integer arithmetic overflow either throws an OverflowException or discards the most significant bits of the result. Integer division by zero always throws a **DivideByZeroException**.
* Floating-point arithmetic overflow or division by zero never throws an exception, because floating-point types are based on IEEE 754 and so have provisions for representing infinity and NaN (Not a Number).
* Decimal arithmetic overflow always throws an OverflowException. Decimal division by zero always throws a DivideByZeroException.

When integer overflow occurs, what happens depends on the execution context, which can be checked or unchecked. In a checked context, an OverflowException is thrown. In an unchecked context, the most significant bits of the result are discarded and execution continues. Thus, C# gives you the choice of handling or ignoring overflow.

In addition to the arithmetic operators, integral-type to integral-type casts can cause overflow, for example, casting a long to an int, and are subject to checked or unchecked execution. However, bitwise operators and shift operators never cause overflow.