Versions:

* ISO-1
* ISO-2
* 3
* 4
* 5
* 6
* 7
* 7.1
* 7.2
* 7.3
* 8

Keywords – słowa zastrzeżone o specjalnym znaczeniu. Można ich używać z @ przed np. @if

Contextual keywords – mają szczególne znaczenie tylko w niektórych miejscach

“At. Generally,as new keywords areadded to the C# language, they areadded as contextual keywords in order to avoid breaking programs written in earlier versions.”

**Operators:**

x?.y – null conditional member access. Returns null if the left-hand operand evaluates to null .

x?[y] - null conditional index access. Returns null if the left-hand operand evaluates to null .

default(T) – produces the default value of typeT.

sizeof – returns the sizein bytes of the type operand

-> – pointer dereferencing combined with member access (unsafe – pointers).

(T)x– typecasting.

&x– address of. (unsafe mode)

\*x– dereferencing.

x / y – division. If the operands are integers, the result is an integer truncated toward zero (for example, -7 / 2 is -3 )

x << y – shift bits left and fill with zero on the right.

Why use shift operators? They don't seem to do much. Well, there are 2 reasons:

1. They are really fast, because nearly all CPUs have shift registers, meaning the shifting operation is done in the hardware, in the minimum amount of effort (cycles).

2. Because they are fast, a lot of protocols and standards are designed to take advantage of this. For example IP address operations, checking a CRC, graphic operations etc.

3. It's useful to write powers of two.

x >> y – shift bits right. If the left operand is int or long , then left bits are filled with the sign bit. If the left operand is uint or ulong , then left bits are filled with zero.

is – typecompatibility. Returns trueif theevaluated left operand can becast to thetypespecified in theright operand (a static type).

as – type conversion. Returns the left operand cast to the type specified by the right operand (a static type), but as returns null where (T)x would throw an exception.

x == y – equality. By default, for reference types other than string , this returns reference equality (identity test). However, types can overload == , so if your intent is to test identity, it is best to use the ReferenceEquals method on object .

x != y – notequal .See comment for == . If a type overloads == , then it must overload != .

x & y, x ^ y, x | y, x – logical AND, XOR, OR for the bool operands or bitwise logical AND for the operands of the integral types

x && y – logical AND. If the first operand evaluates to false, then C# does not evaluate the second operand.

x += y, x -= y, x \*= y, x /= y, x %= y, x &= y, x |= y, x ^= y, – increment. Add the value of y to the value of x , store the result in x ,and return the new value. If x designates an event , then y must bean appropriate function that C# adds as an event handler.

x <<= y, x >>= y – left-shift assignment. Shift the value of x left by y places, store the result in x ,and return the new value.

x ?? y – (Null-coalescing operator) returns x if it is non- null ; otherwise, returns y .

t ? x :y – Conditional operator if test t evaluates to true, then evaluateand return x ; otherwise,evaluateand return y .

**Special Characters**

**@**, theverbatim identifier character

* To enable C# keywords to be used as identifier. @for, @int
* To indicate that a string literal is to be interpreted verbatim (escapes \\ in string)
* Only a quote escape sequence ("") is not interpreted literally; it produces a single quotation mark
* in case of a verbatim [interpolated string](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated) brace escape sequences ({{ and }}) are not interpreted literally; they produce single brace characters
* To enable the compiler to distinguish between attributes in cases of a naming conflict.
  + public class Info : Attribute
  + public class InfoAttribute : Attribute

[Info("A simple executable.")] // Generates compiler error CS1614. Ambiguous Info and InfoAttribute.

// Prepend '@' to select 'Info'. Specify the full name 'InfoAttribute' to select it.

* + public class Example

$, theinterpolated string character

{<interpolatedExpression>[,<alignment>][:<formatString>]}

| **Element** | **Description** |
| --- | --- |
| interpolatedExpression | The expression that produces a result to be formatted.  String representation of the null result is [String.Empty](https://docs.microsoft.com/en-us/dotnet/api/system.string.empty). |
| alignment | The constant expression whose value defines the  minimum number of characters in the string representation  of the result of the interpolated expression. If positive,  the string representation is right-aligned; if negative,  it's left-aligned. For more information, see [Alignment Component](https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting#alignment-component). |
| formatString | A format string that is supported by the type of the  expression result. For more information,  see [Format String Component](https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting#format-string-component). |

To include a brace, "{" or "}", in the text produced by an interpolated string, use two braces, "{{" or "}}"

The $ token must appear before the @ token in a verbatim interpolated string.

As the colon (":") has special meaning in an interpolated expression item, in order to use a [conditional operator](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator) in an interpolated expression, enclose that expression in parentheses.