

# Operator Precedence

**Operator precedence** determines the order in which the operators in an expression are evaluated.

For eg –

`int x = 3 * 4 - 1;`

In the above example, the value of x will be 11, not 9. This happens because the precedence of `*` operator is higher than `-` operator. That is why the expression is evaluated as  $(3 * 4) - 1$  and not  $3 * (4 - 1)$ .

## Operator Precedence Table

Operators	Precedence
postfix increment and decrement	<code>++</code> <code>--</code>
prefix increment and decrement, and unary	<code>++</code> <code>--</code> <code>+</code> <code>-</code> <code>~</code> <code>!</code>
multiplicative	<code>*</code> <code>/</code> <code>%</code>
additive	<code>+</code> <code>-</code>
shift	<code>&lt;&lt;</code> <code>&gt;&gt;</code> <code>&gt;&gt;&gt;</code>
relational	<code>&lt;</code> <code>&gt;</code> <code>&lt;=</code> <code>&gt;=</code> <code>instanceof</code>
equality	<code>==</code> <code>!=</code>
bitwise AND	<code>&amp;</code>
bitwise exclusive OR	<code>^</code>
bitwise inclusive OR	<code> </code>
logical AND	<code>&amp;&amp;</code>
logical OR	<code>  </code>
ternary	<code>? :</code>
assignment	<code>=</code> <code><code>i</code>=</code> <code><code>+=</code></code> <code><code>*</code>=</code> <code><code>/=</code></code> <code><code>%=</code></code> <code><code>&amp;=</code></code> <code><code>^=</code></code> <code><code> =</code></code> <code><code>&lt;&lt;=</code></code> <code><code>&gt;&gt;=</code></code> <code><code>&gt;&gt;&gt;=</code></code>

## Associativity of Operators

shelkeomkar272@gmail.com

If an expression has two operators with similar precedence, the expression is evaluated according to its **associativity** (either left to right, or right to left).

Operators	Precedence	Associativity
postfix increment and decrement	<code>++</code> <code>--</code>	left to right
prefix increment and decrement, and unary	<code>++</code> <code>--</code> <code>+</code> <code>-</code> <code>~</code> <code>!</code>	right to left
multiplicative	<code>*</code> <code>/</code> <code>%</code>	left to right
additive	<code>+</code> <code>-</code>	left to right
shift	<code>&lt;&lt;</code> <code>&gt;&gt;</code> <code>&gt;&gt;&gt;</code>	left to right
relational	<code>&lt;</code> <code>&gt;</code> <code>&lt;=</code> <code>&gt;=</code> <code>instanceof</code>	left to right
equality	<code>==</code> <code>!=</code>	left to right
bitwise AND	<code>&amp;</code>	left to right
bitwise exclusive OR	<code>^</code>	left to right
bitwise inclusive OR	<code> </code>	left to right
logical AND	<code>&amp;&amp;</code>	left to right
logical OR	<code>  </code>	left to right
ternary	<code>? :</code>	right to left
assignment	<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code>&amp;=</code> <code>^=</code> <code> =</code> <code>&lt;&lt;=</code> <code>&gt;&gt;=</code> <code>&gt;&gt;&gt;=</code>	right to left

**Note -** These notes are just for a quick glance. We don't have to memorize them all at once. Most of these rules are very logical and we have been following them in a lot of instances already.

Shelkeomkar272@gmail.com