Operator Precedence

- Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.
    - ➔ Dictates the order of evaluations when two operators share an operand.
    - ➔ Certain operators have higher precedence than others.
    - ➔ For example, the multiplication operator has a higher precedence than the addition operator.

x= 7+3*2;

- Can result in 13 or 20 depending on the order of each operands evaluation.
- The order of executing the various operations can make a difference , so C needs unambiguous rules for choosing what to do first.
- In C , x is assigned 13, not 20 because operator * has a higher precedence than +.
    - ➔ Firsts gets multiplied with 3*2 and then adds into 7
- Each operator is assigned a precedence level.
    - ➔ Multiplication and division have a higher precedence than addition and subtraction, so they ate performed first.
- Whatever is enclosed in parentheses is executed first, should just always use () to group expressions.

Associativity

`the **associativity** of an operator **is** a property that determines how operators of the same precedence **are** grouped in the absence of parentheses.`

- What if two operators have the same precedence?
    - ➔ Then associativity rules are applied.
- If they share an operand, they executed according to the order in which they occur in the statement,
    - ➔ For most operators, the order is from left to right.

1 == 2 !=3

- Operators == and != have same precedence
    - ➔ Associativity of both == and != is left to right .
    - ➔ The expression on the left is executed first and moves towards the right.
- The expression above is equivalent to

((1 ==2) != 3)

- (1 == 2) executes first resulting into 0 (false) , then , (0!=3) executes resulting into 1 (true)
-

Table (highest to lowest) (tutorials point)

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right

Table (highest to lowest) (tutorials point)

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right