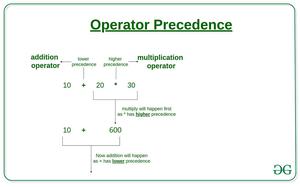
**Operator Precedence and Associativity**

Operator Precedence determines which operation is performed first in an expression with more than one operators with different precedence.  
**For example:** Solve

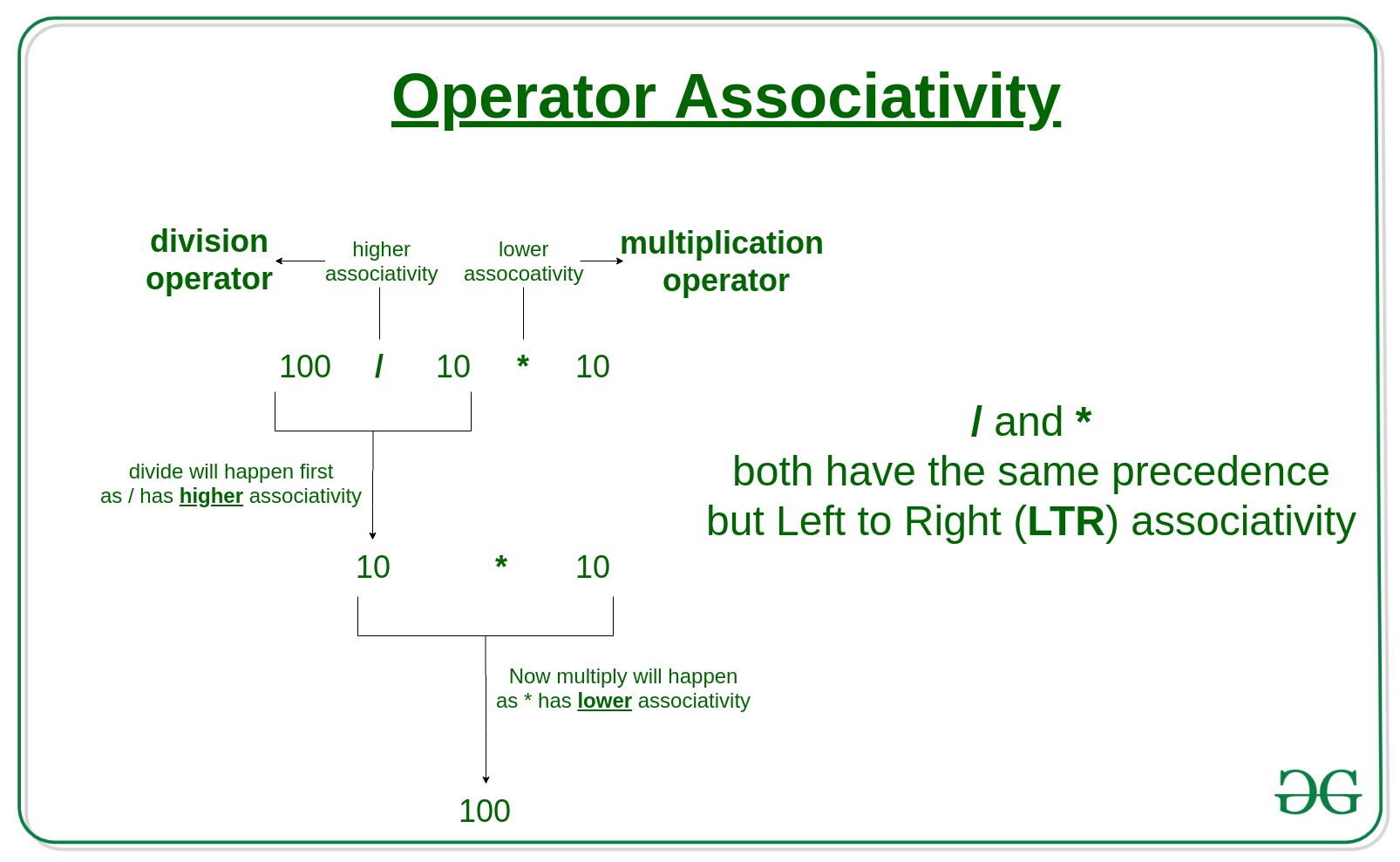
10 + 20 \* 30



**10 + 20 \* 30 is calculated as 10 + (20 \* 30)**

and not as **(10 + 20) \* 30**

**Operators Associativity** is used when two operators of same precedence appear in an expression. Associativity can be either Leftto Right orRightto Left.   
**For example:** ‘\*’ and ‘/’ have same precedence and their associativity is Leftto Right, so the expression “100 / 10 \* 10” is treated as “(100 / 10) \* 10”.

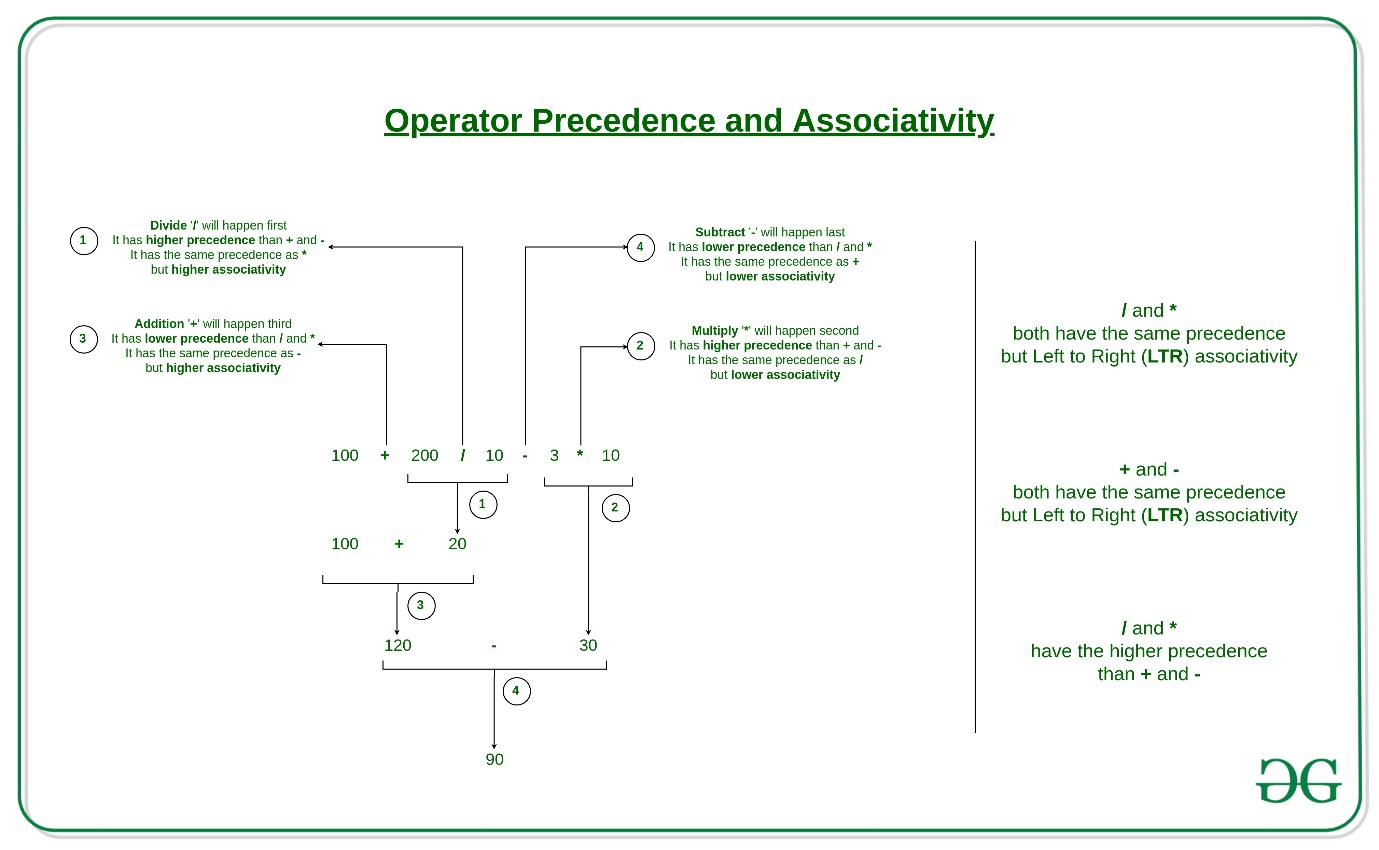


***Operators Precedence and Associativity are two characteristics of operators***

***that determine the evaluation order of sub-expressions in absence of brackets.***

**For example:** Solve

100 + 200 / 10 - 3 \* 10



**1) Associativity is only used when there are two or more operators of same precedence.**   
**2) All operators with the same precedence have same associativity**

**3) Precedence and associativity of postfix ++ and prefix ++ are different**

**4) Comma has the least precedence among all operators and should be used carefully**

**5) There is no chaining of comparison operators**

**Example 1: Operators Precedence**

C++

#include <iostream>

using namespace std;

int main() {

// evaluates 17 \* 6 first

int num1 = 5 - 17 \* 6;

// equivalent expression to num1

int num2 = 5 - (17 \* 6);

// forcing compiler to evaluate 5 - 17 first

int num3 = (5 - 17) \* 6;

cout << "num1 = " << num1 << endl;

cout << "num2 = " << num2 << endl;

cout << "num3 = " << num3 << endl;

return 0;

}

**Output**

num1 = -97

num2 = -97

num3 = -72

**Example 2: Operators Associativity**

C++

#include <iostream>

using namespace std;

int main() {

int a = 1;

int b = 4;

// a -= 6 is evaluated first

b += a -= 6;

cout << "a = " << a << endl; ;

cout << "b = " << b;

}

**Output**

a = -5

b = -1

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Associativity** |
| **( )  [ ]  .  ->  ++ –  –** | **Parentheses (function call) (see Note 1)  Brackets (array subscript)  Member selection via object name  Member selection via pointer  Postfix increment/decrement (see Note 2)** | **left-to-right** |
| **++ –  –  + –  ! ~  (*type*)  \*  &  sizeof** | **Prefix increment/decrement  Unary plus/minus  Logical negation/bitwise complement  Cast (convert value to temporary value of *type*)  Dereference  Address (of operand)  Determine size in bytes on this implementation** | **right-to-left** |
| **\*  /  %** | **Multiplication/division/modulus** | **left-to-right** |
| **+  –** | **Addition/subtraction** | **left-to-right** |
| **<<  >>** | **Bitwise shift left, Bitwise shift right** | **left-to-right** |
| **<  <=  >  >=** | **Relational less than/less than or equal to  Relational greater than/greater  than or equal to** | **left-to-right** |
| **==  !=** | **Relational is equal to/is not equal to** | **left-to-right** |
| **&** | **Bitwise AND** | **left-to-right** |
| **^** | **Bitwise exclusive OR** | **left-to-right** |
| **|** | **Bitwise inclusive OR** | **left-to-right** |
| **&&** | **Logical AND** | **left-to-right** |
| **| |** | **Logical OR** | **left-to-right** |
| **? :** | **Ternary conditional** | **right-to-left** |
| **=  +=  -=  \*=  /=  %=  &=  ^=  |=  <<=  >>=** | **Assignment  Addition/subtraction assignment  Multiplication/division assignment  Modulus/bitwise AND assignment  Bitwise exclusive/inclusive OR assignment  Bitwise shift left/right assignment** | **right-to-left** |
| **,** | **Comma (separate expressions)** | **left-to-right** |

It is good to know precedence and associativity rules, but the best thing is to use brackets, especially for less commonly used operators (operators other than +, -, \*.. etc). Brackets increase the readability of the code as the reader doesn’t have to see the table to find out the order.