Covered Topics Under UNIT-2 of "PPS-PROGRAMMING FOR PROBLEM SOLVING (BCS101 / BCS201)"

shwetatiwari08@recabn.ac.in
shwetatiwari08aug@gmail.com

Class Notes

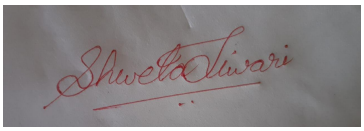
Published Date: November, 2022

PPS: UNIT-2

Arithmetic Expression and Precedence

FALL SEMESTER, YEAR (I/II sem, 1st yr)

FALL SESSION (2022-23)
(PPS)
MS. SHWETA TIWARI
Published Date: November, 2022



shwetatiwari08@recabn.ac.in
shwetatiwari08aug@gmail.com

TOPIC On : UNIT-2: Operator Precedence and Associativity in C

—

By SHWETA TIWARI

Under On: Arithmetic Expression and Precedence

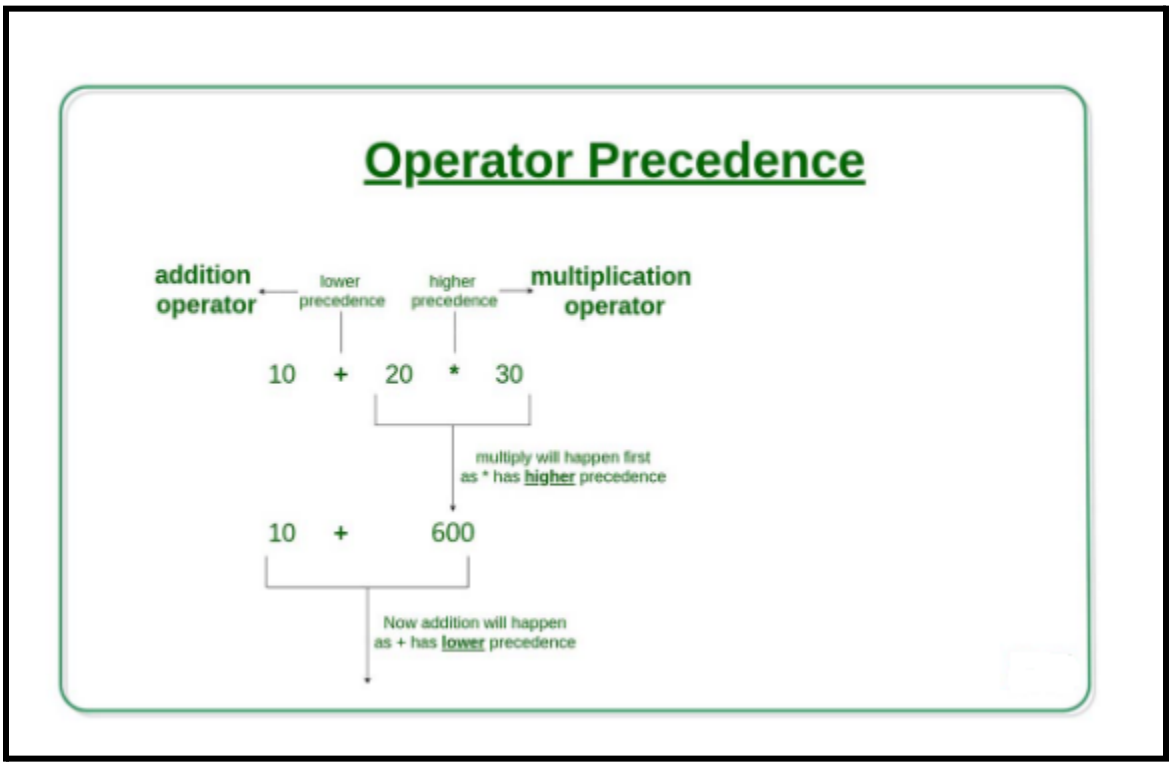
PREPARED FOR
Engineering Students
All Engineering College

PREPARED BY
SHWETA TIWARI

TOPIC On : ***UNIT-2:*** Operator Precedence and Associativity in C

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

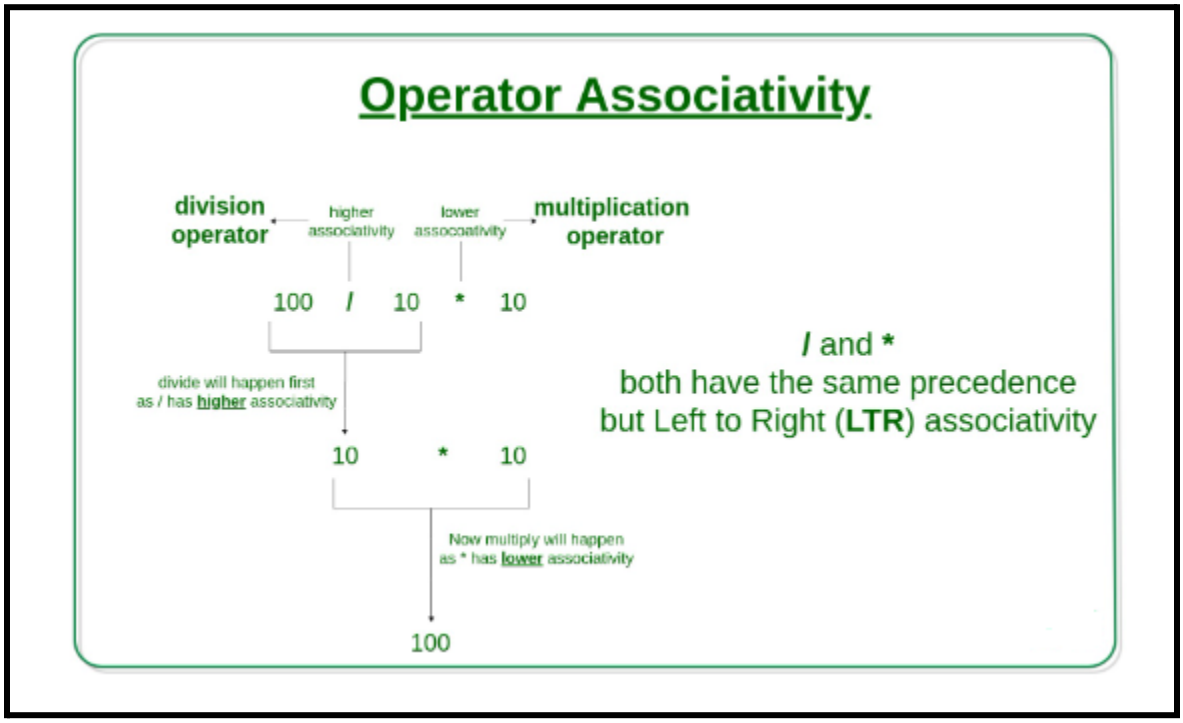
10 + 20 * 30



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 **L**eft **t**o **R**ight or **R**ight **t**o **L**eft.

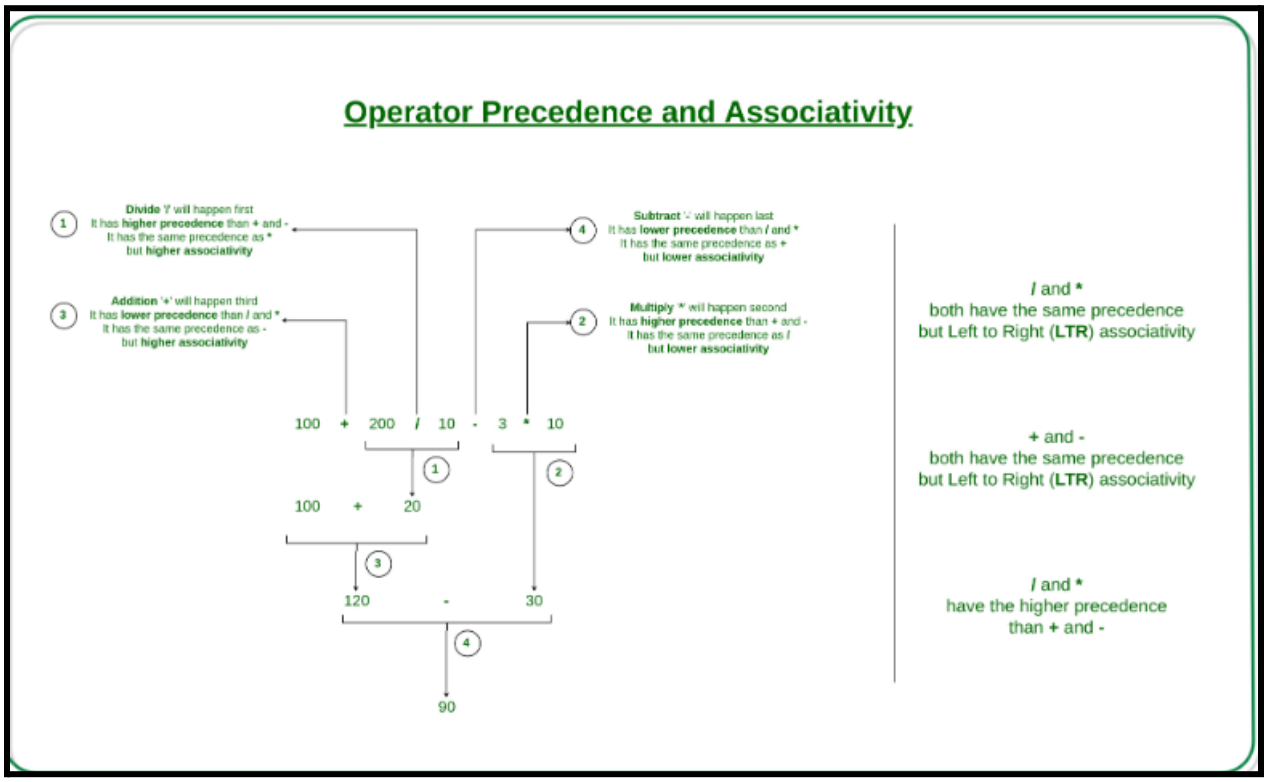
For example: '*' and '/' have same precedence and their associativity is **L**eft **t**o **R**ight, so the expression "100 / 10 * 10" is treated as "(100 / 10) * 10".



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 the same precedence.

The point to note is that associativity doesn't define the order in which operands of a single operator are evaluated. For example, consider the following program, associativity of the + operator is left to right, but it doesn't mean f1() is always called before f2(). The output of the following program is in-fact compiler dependent. See this for details.

2) All operators with the same precedence have same associativity

This is necessary, otherwise, there won't be any way for the compiler to decide the evaluation order of expressions which have two operators of the same precedence and different associativity. For example + and – have the same associativity.

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

Precedence of postfix ++ is more than prefix ++, their associativity is also different. Associativity of postfix ++ is left to right and associativity of prefix ++ is right to left. See this for examples.

4) Comma has the least precedence among all operators and should be used carefully For example consider the following program, the output is 1. See this and this for more details.

5) There is no chaining of comparison operators in C

In Python, expressions like "c > b > a" are treated as "c > b and b > a", but this type of chaining doesn't happen in C. For example, consider the following program. The output of the following program is "FALSE".

Use of the Operator Precedence and Associativity in C

Precedence and Associativity are two of the characters that are used for operators in an expression for determining the order of sub-expressions when they do not have brackets.

Summarize that:

1. **Operator Precedence**

Operator precedence helps us determine which of the operators in an expression must be evaluated first in case the expression consists of more than a single operator.

2. **Operator Associativity**

We use associativity when two or more than two operators with the same precedence are present in the same expression.

C Operator Precedence Table

C operators are listed in order of *precedence* (highest to lowest). Their *associativity* indicates in what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity
() [ ] . -> ++ --	Parentheses: grouping or function call Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	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