Operator Precedence and Associativity in C

Operator precedence determines which operator 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

= 100 + (200 / 10) – (3 * 10) = 100 + 20 – 30 = (100 + 20) – 30

=120 – 30 = 90

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
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.

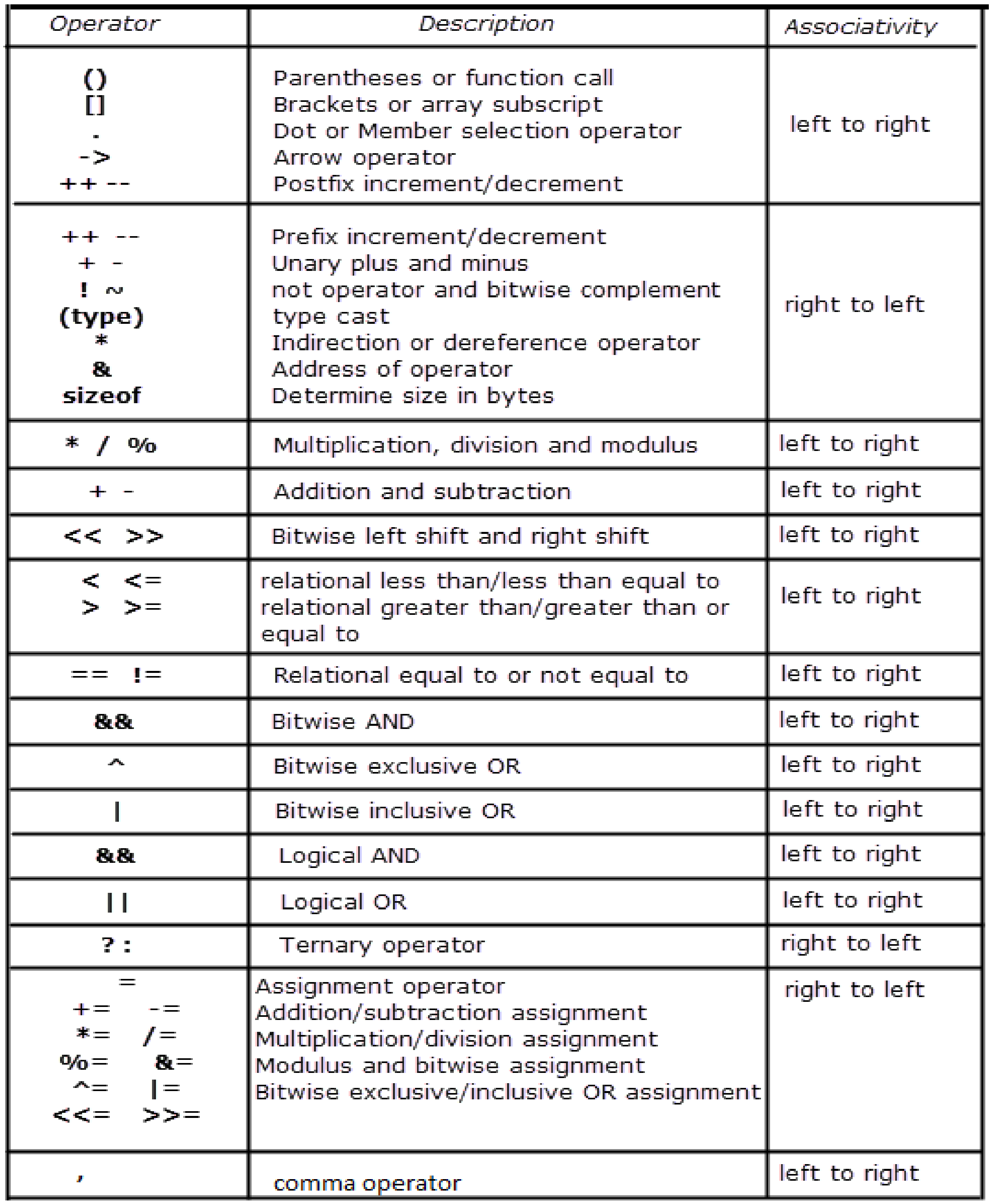
4) Comma has the least precedence among all operators and should be used carefully.

5) There is no chaining of comparison operators in C
In Python, expression like "c > b > a" is 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 following program is "FALSE".

```
#include <stdio.h>
int main()
{
    int a = 10, b = 20, c = 30;
    // (c > b > a) is treated as ((c > b) > a), associativity of '>'
    // is left to right. Therefore, the value becomes ((30 > 20) > 10)
    // which becomes (1 > 20)
    if (c > b > a)
        printf("TRUE");
    else
```

```
        printf("FALSE");
    return 0;
}
Output: FALSE
```

Please see the following precedence and associativity table for reference.



Operator	*Description*	*Associativity*
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to or 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 operator	right to left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	comma operator	left to right