Precedence and Associativity in C Programming

Operator precedence describes the order in which C reads expressions. For example, the expression `a=4+b*2` contains two operations, an addition and a multiplication. Does the C compiler evaluate `4+b` first, then multiply the result by `2`, or does it evaluate `b*2` first, then add `4` to the result? The operator precedence chart contains the answers. Operators higher in the chart have a higher precedence, meaning that the C compiler evaluates them first. Operators on the same line in the chart have the same precedence, and the "Associativity" column on the right gives their evaluation order.

Precedence	Operator	Description	Associativity
1	value++ value--	postfix increment and decrement	left - to - right
2	++value --value + - ! ~ (type) & sizeof()	prefix increment and decrement unary plus and unary minus logical NOT and bitwise NOT typecast address of size of	right - to left
3	* / %	multiplication, division and remainder	left - to - right
4	+ -	addition and subtraction	left - to - right
5	<< >>	bitwise left shift, bitwise right shift	left - to - right
6	< <= > >=	less than, less than or equal to greater than, greater than or equal to	left - to - right
7	== !=	relational equal to and not equal to	left - to - right
8	&	bitwise AND	left - to - right
9	^	bitwise XOR (exclusive OR)	left - to - right
10	|	bitwise OR (inclusive OR)	left - to - right
11	&&	logical AND	left - to - right
12	||	logical OR	left - to - right
13	? :	ternary operator	right - to - left
14	= += -= *= /= %= <<= >>= &= ^= |=	simple assignment assignment with sum and difference assignment with sum, quotient and remainder assignment with bitwise left shift and right shift assignment with bitwise AND, XOR and OR	right - to - left
15	,	comma	left - to - right

Precedence and associativity are independent from [order of evaluation].

The C language standard doesn't specify operator precedence. It specifies the language grammar, and the precedence table is derived from it to simplify understanding. There is a part of the grammar that cannot be represented by a precedence table: an assignment-expression is not allowed as the right hand operand of a conditional operator, so `e = a < d ? a++ : a = d` is an expression that would not compile, and therefore relative precedence of conditional and assignment operators cannot be described easily.

1) Associativity is only used when there are two or more operators of same precedence. The point to note is associativity doesn't define the order in which operands of a single operator are evaluated.

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

For example consider the following program, the output is 1.

```
#include<stdio.h>
int main()
{
    int a;
    a = 1, 2, 3;
    // Evaluated as (a = 1), 2, 3
    printf("%d", a);
    return 0;
}
```

5) There is no chaining of comparison operators in C In Python, expression like "c > b > a" is treated as "a > b and b > c", 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;
}
```