0301102 LOGIC DEVELOPMENT & PROGRAMMING

UNIT – 2

Principles of Programming Language

Operators & Expressions

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- ***Consider the expression A + B * 5. where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.***

Operators

- C language supports a rich set of built-in operators. **An operator is a symbol that tells the compiler to perform a certain mathematical or logical operations.**
- Operators are used in programs to manipulate data and variables.
- **An operator is a symbol that operates on a value or a variable. For example: + is an operator to perform addition.**
- **Unary Operator:** A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.
- **Binary Operator:** The binary operator is an operator applied between two operands. The following is the list of the binary operators:

Operators

- **C operators can be classified into following types:**
  - **Arithmetic operators**
  - **Relational operators**
  - **Logical operators**
  - **Bitwise operators**
  - **Assignment operators**
  - **Conditional operators**
  - **Special/Misc operators**

Operators

- **Unary operators:** It has two operators increment ++ and decrement -- to change the value of an operand (varible) by 1.
- Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, that is it only operate on a single operand.
- **Syntax:**
  - Increment operator: ++var_name; (or) var_name++;
  - Decrement operator: – -var_name; (or) var_name – -;
- **Example:**
  - Increment operator : ++ i ; i ++ ;
  - Decrement operator : – – i ; i – – ;

Operators

- **Prefix and Postfix in C**
- If you observe the above syntax, we can assign the C increment and decrement operators either before operand or after the operand. When ++ or — is used before operand like: ++x, –x then we call it as prefix, if ++ or — is used after the operand like: x++ or x– then we called it as postfix.
- Let’s explore the prefix and postfix of increment and decrement operators in C:
  - ++i (Pre-increment): It will increment the value of i even before assigning it to the variable i.
  - i++ (Post-increment): The operator will return the variable value first (i.e, i value) then only i value is incremented by 1.
  - –i (Pre decrement): It will decrement the value of i even before assigning it to the variable i.
  - i– (Post decrement): The operator returns the variable value first (i.e., i value), then only i value decremented by 1.

Operators

- **Arithmetic operators:** An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values.

subtraction or unary minus
multiplication
division

Operators

- **Assignment Operators:** An assignment operator is used for assigning a value to a variable. The most common assignment operator is =.

a += b	a = a
a -= b	a = a
a *= b	a = a
a /= b	a = a
a %= b	a = a

Operators

- **Relational Operators:** A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.

Greater than	`5 > 3` is evaluated to 1
Less than	`5 < 3` is evaluated to 0
Not equal to	`5 != 3` is evaluated to 1
Greater than or equal to	`5 >= 3` is evaluated to 1
Less than or eaual to	`5 <= 3` is evaluated to 0

Operators

- **Logical Operators:** An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.
- Logical operators are commonly used in decision making in C programming.

operands are true	((c==5) && (d>5)) equals to 0.
Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
Logical NOT. True only if the	If c = 5 then, expression !(c==5) equal

Operators

- **Bitwise Operators:** Bitwise operators perform manipulations of data at bit level. These operators also perform shifting of bits from right to left.
- Bitwise operators are not applied to float or double.

1. The **& (bitwise AND)** in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. The **| (bitwise OR)** in C takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
3. The **^ (bitwise XOR)** in C takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
4. The **<< (left shift)** in C takes two numbers, the left shifts the bits of the first operand, and the second operand decides the number of places to shift.
5. The **>> (right shift)** in C takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.
6. The **~ (bitwise NOT)** in C takes one number and inverts all bits of it.

Let's look at the truth table of the bitwise operators.

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Operators

- **Bitwise AND operator &:** The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.
- Let us suppose the bitwise AND operation of two integers 12 and 25:

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

00001100

& 00011001

00001000 = 8 (In decimal)

Operators

- **Bitwise OR operator | :** The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

```
  00001100
| 00011001
  00011101  = 29 (In decimal)
```

Operators

- **Bitwise XOR (exclusive OR) operator ^ :** The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

00001100

^ 00011001

00010101 = 21 (In decimal)

Operators

- **Bitwise Operators:** The bitwise shift operator, shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value have to be shifted. Both operands have the same precedence.

Operators

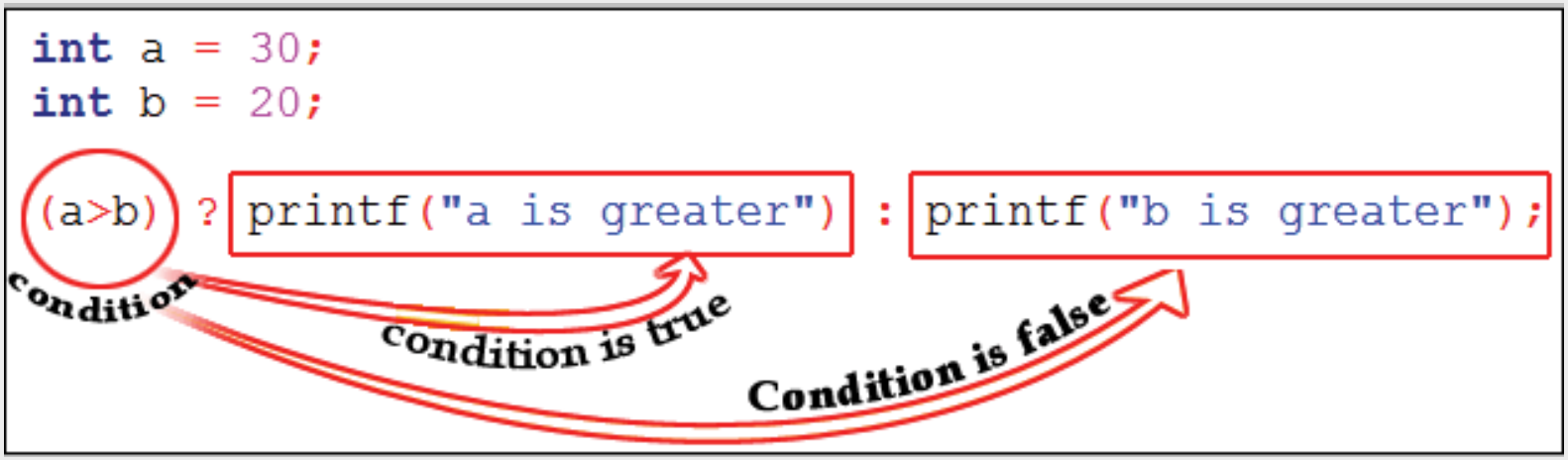
- **Bitwise complement operator ~ :** Bitwise complement of any number N is -(N+1)
- **Shift Operators in C programming**
- There are two shift operators in C programming:
  - Right shift operator
  - Left shift operator.
- **Right Shift Operator**
- Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.
  - 212 = 11010100 (In binary)
  - 212>>2 = 00110101 (In binary) [Right shift by two bits]
  - 212>>7 = 00000001 (In binary)
  - 212>>8 = 00000000
  - 212>>0 = 11010100 (No Shift)

Operators

- **Left Shift Operator:**
- Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with 0. The symbol of the left shift operator is <<.
  - 212 = 11010100 (In binary)
  - 212<<1 = 110101000 (In binary) [Left shift by one bit]
  - 212<<0 = 11010100 (Shift by 0)
  - 212<<4 = 110101000000 (In binary) =3392(In decimal)

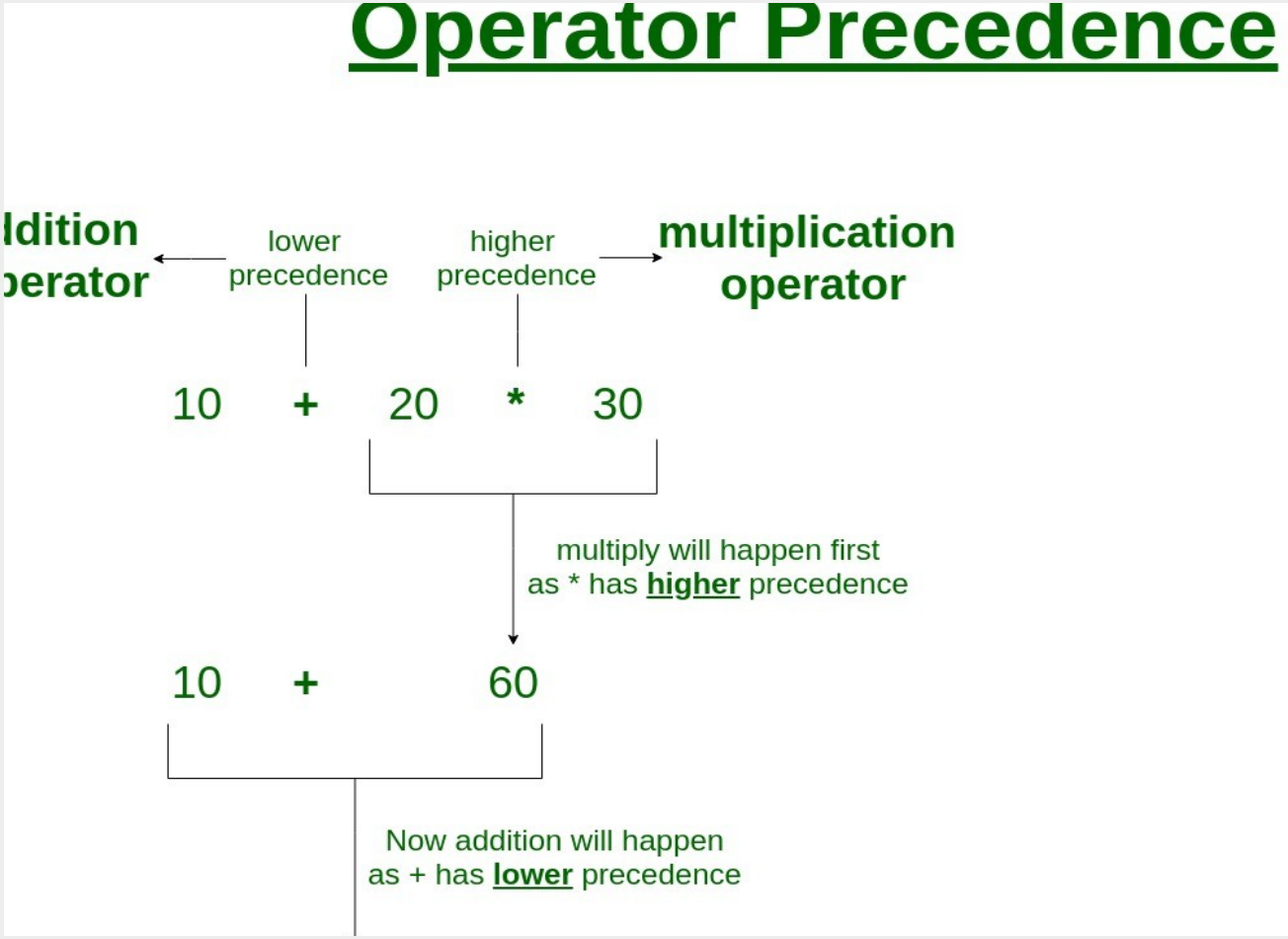
Operators

- **Conditional Operators:** The conditional operators in C language are known by two more names
  - **Ternary Operator**
  - **? : Operator**
- It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.
- The syntax of a conditional operator is :
  - expression 1 ? expression 2: expression 3



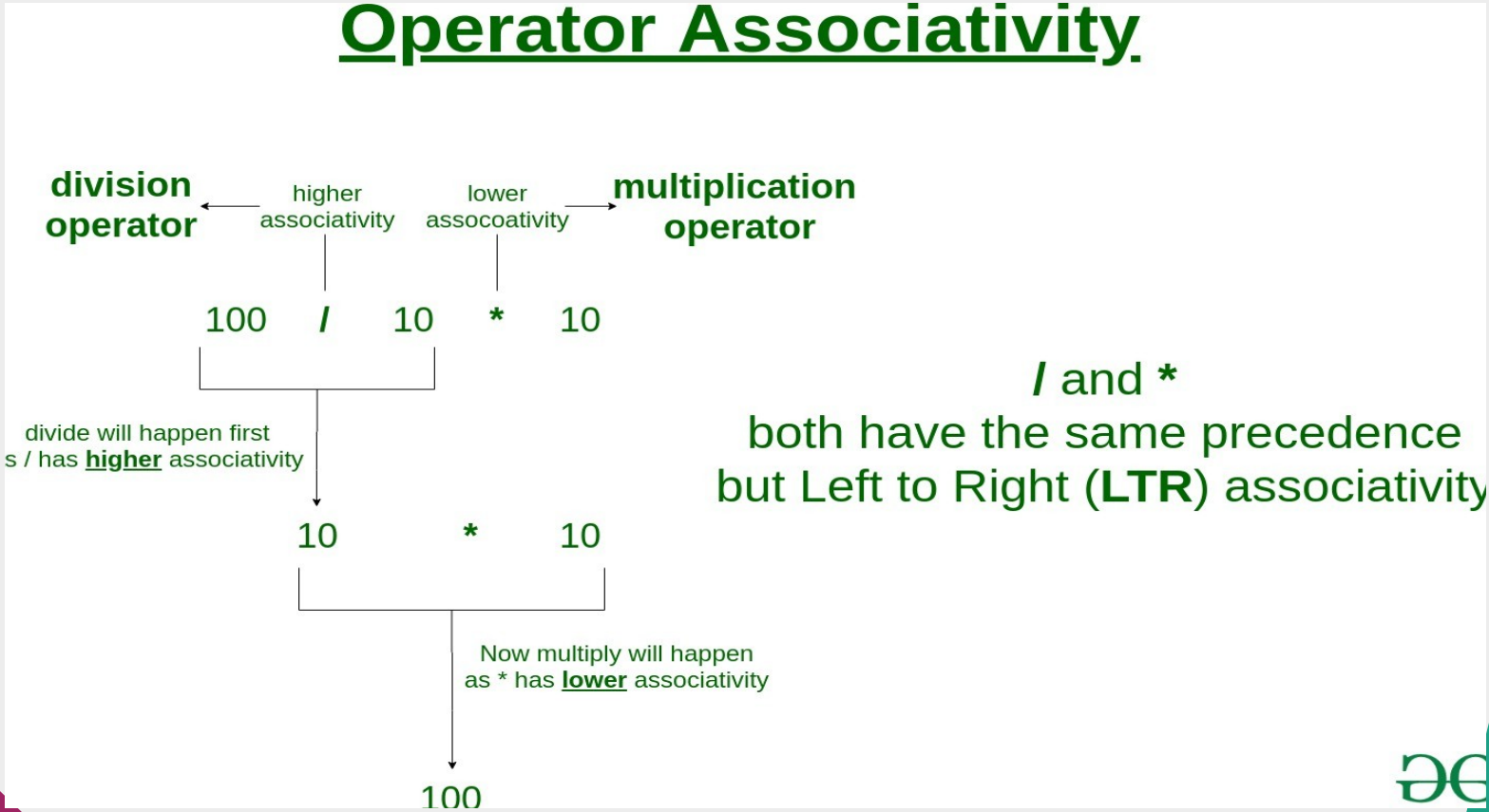
Operator Precedence and Associativity in C

- **Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence.**
- **For example: Solve - 10 + 20 * 30**



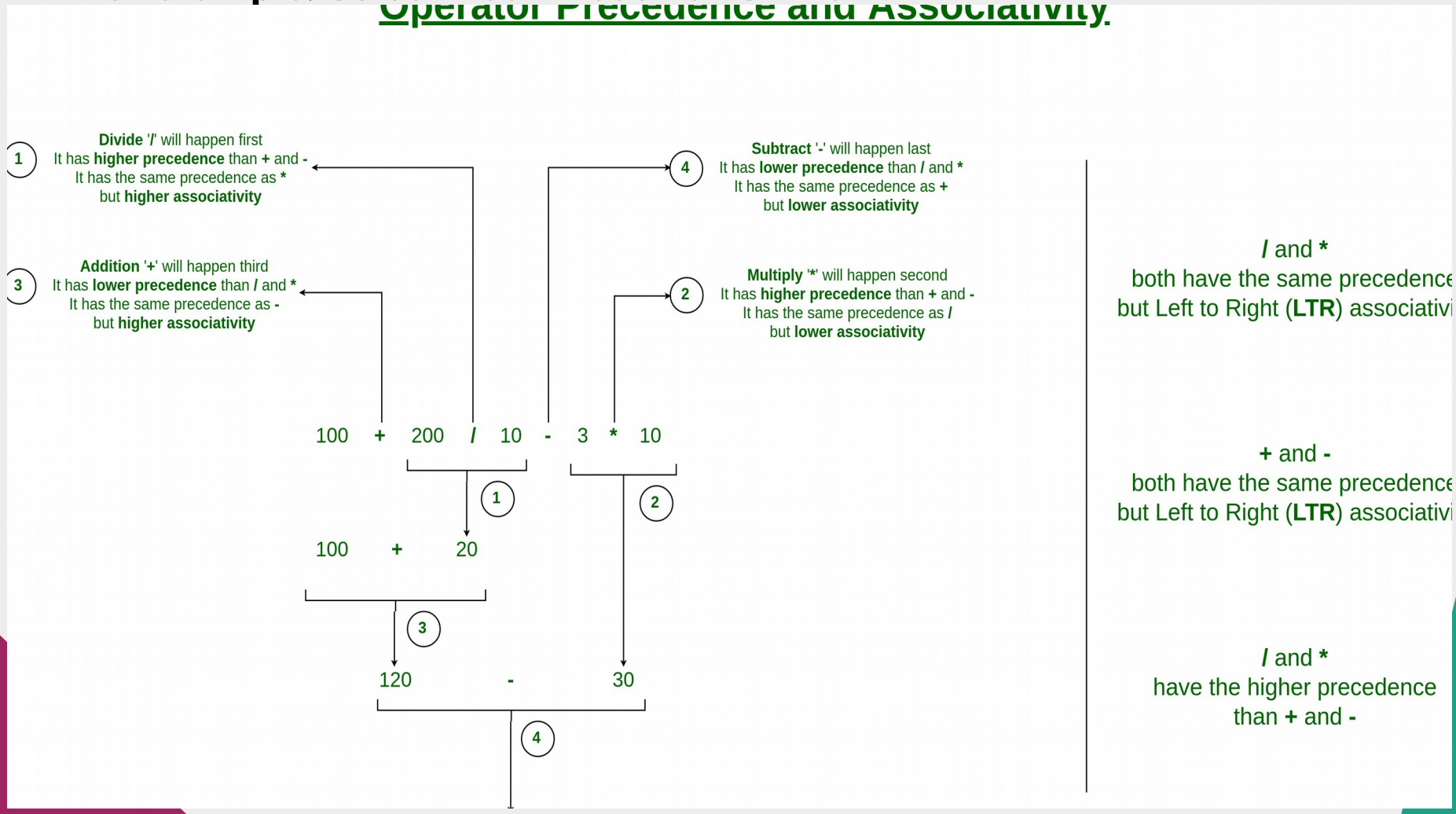
Operator Precedence and Associativity in C

- 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 Left to Right or Right to Left.
- For example: '*' and '/' have same precedence and their associativity is Left to Right, so the expression "100 / 10 * 10" is treated as "(100 / 10) * 10".

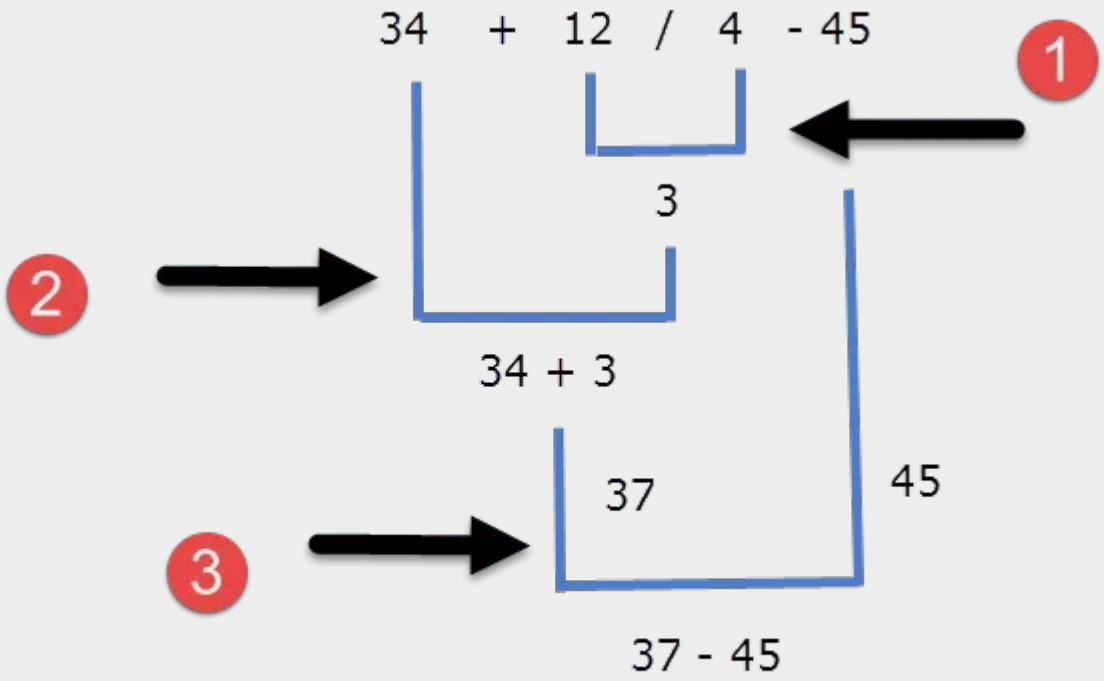


Operator Precedence and Associativity in C

- 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**



Operator Precedence and Associativity in C



Ans : -8

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