Operators in C.

- Operators are ~~the~~ one of the most important part of any C program. These are symbols that help us in performing certain tasks ~~for~~ on the variables, which are called operands. They are 3 types of operators when grouped according to the number of ~~operators to~~ operands required:

(a) Unary Operator : ++, -- (They require one operand)

(b) Binary operators : <, <=, >, >=, ==, !=, &&, ||, !, &, |, <<, >>, ~, ^, =, +=, -=, *=, /=, %=, +, -, *, /, %.
(They require 2 operands)

(C) Ternary operator : ?: (They require 3 operands & is also called Conditional operator)

- The unary operators either increase the value of the variable by 1 (++) or decrease it by 1 (--). When do they ~~do this~~ ~~the~~ change depends on the position of the variable. If the variable is placed before the operator, i.e, a++ or a--, then the value is used before changing. This is called postfix increment / decrement. For example,

```
int a = 1;
int b = a++;  // b=1
int c = a;    // c=2.
```

If the variable is placed after the operator, i.e, ++a or --a, then the value is changed before using it. This is

CLASSMATE
Date
Page

(*) The output is either a true or a false.

called prefix increment / decrement. For example,

```
int a =2;
int b = --a; //b=1
int c = a; //c=1
```

· The binary operators may further be classified into 5 different types:

(i) Arithmetic operators - They are used to perform everyday mathematical operations on the operands, for example + (addition), - (subtraction), * (multiplication), / (finding quotient), % (finding remainder).

(ii) Relational ~~B~~ operators - They are used to compare 2 ~~&~~ operands to find the greater or lesser or equal to. For example, > (greater than), < (less than), >= (greater than or equal to) & <= (lesser than or equal to), == (equal to), != (not equal to)

(iii) Logical Operator - These are used when ~~two~~ we are ~~to~~ trying to impose one ~~any~~ condition on another one, i.e, ~~the~~ when we want two conditions to be satisfied at the same time. (*) There are 3 logical ~~opera~~ operators, (a) && - It is a representation of logical ~~&&~~ and. If we write a&&b then this would return true ONLY IF both a and b are true, otherwise false will be returned. (b) || - It is a representative of the logical OR. If we write a||b then this would return ~~fa~~ true IF ANY of ~~the~~ a or b is true, otherwise false will be returned. (c) ! - It's a representative of the logical NOT. If we write !a then it would return false if a is true and vice-versa.

(iv) Bitwise Operators -

They are used when we want to perform some bit-level operations on the operands. When the mathematical calculation happens at the bit-level they are faster than in the normal level. Here both the operands are converted to bit-level and then the operation is performed. For example, & (AND), | (OR), ^ (XOR), << (left shift), >> (right shift) & ~ (NOT).

(v) Assignment Operators -

They are used to assign different values to the operands. Unlike the other binary operators one of the operands here is a VALUE, i.e; Variable Assignment operator VALUE. The value must be of the same data-type as the variable. For example, = (equal to), += (similar to a = a + x where x is a value), -= (similar to a = a - x), *= (similar to a = a * x), /= (similar to a = a/x)

- The conditional or ternary operator, ?:, is written of the form Expression1 ? Expression2 : Expression3. If Expression1 is true then Expression 2 will be executed else expression3 will be executed.

- All the operators mentioned above have a certain order in which they work. This order is given by the OPERATOR PRECEDENCE CHART, given in table 3.

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	( )	Function call	
	[ ]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right