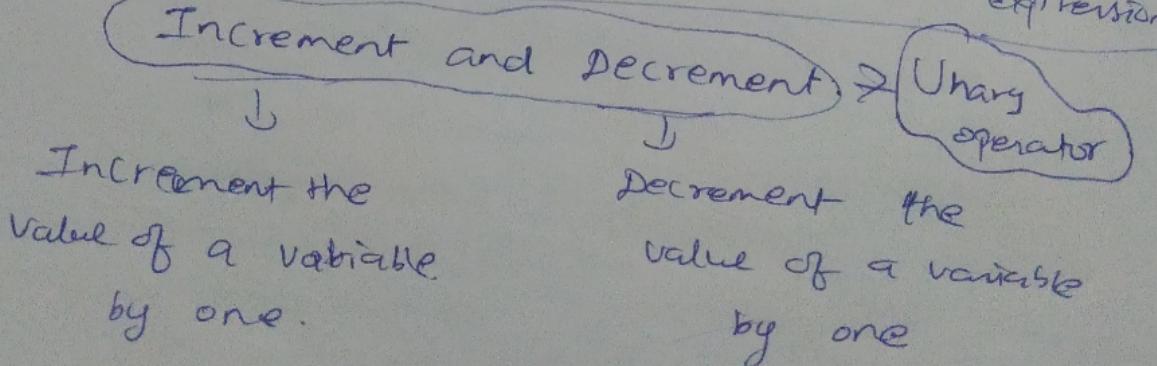


## Introduction to 'C' program - operator

- Arithmetic operator (+, -, /, \*, %)
- Relational operator ( $>=$ ,  $<=$ ) ( $=$ ) ( $!=$ ) ( $<$ ,  $>$ )
- logical operator ( $\&$ ,  $||$ ,  $\sim$ )
- Increment, Decrement ( $++$ ,  $--$ )
- Bitwise operator ( $\&$ ,  $\wedge$ ,  $\mid$ ,  $\sim$ ,  $>>$ ,  $<<$ )
- Assignment operator }  $\Rightarrow$   $=$ ,  $+=$ ,  $-=$ ,  $/=$ ,  
 $%=$ ,  $<<=$ ,  $>>=$ ,  $\&=$ ,  
 $\wedge=$ ,  $\mid=$
- other operator  
[ ?: & \* size of ( ), ]

### ARITHMETIC OPERATOR

- + , - , \* , / , %
- all are binary operator → They need "2" operands.
- Operator Precedence and Associativity
  - Highest \* , / , % - left to right
  - Lower + , - - left to right
- 2 or more operators are of same expression.



```
int a=5  
a++;
```

a=6

```
int a=5  
a--;  
a=4
```

$a = 5$

$$a++; \Rightarrow a = a + 1 = 5 + 1 = 6$$

$$a--; \Rightarrow a = a - 1 = 5 - 1 = 4$$

2 types  $\leftarrow$  Pre  
Post

→ Pre-increment

$++a;$

$$\begin{array}{c} x \\ \boxed{6} \end{array} = \boxed{\underline{7}} \begin{array}{c} 6 \\ a \end{array}$$

Post-increment

→ Pre decrement

$--a;$

→ Post decrement

$a++; x a$

$a--;$

$$\begin{array}{c} \boxed{5} \\ n \end{array} = \boxed{\underline{6}} \begin{array}{c} 6 \\ a \end{array}$$

Ivalue → left value → object that has an  
 ↓  
 identifiable location in memory.  
 must be a } (i.e having an address),  
variable. } because they have the  
 capability to store the data.

Ivalue - can't be a function.

Rvalue → no identifiable location in  
 memory

↳ anything which is capable of returning a  
 constant expression or value.

→ it may be a (expression)

→ it's not a variable,

## Token generation

→ lexical analysis is the first phase  
in compilation process  
like a scanner.

It scans the source program to meaningful sequence of characters.  
↓  
Token  
↓  
mapped into token-name and attribute-value

example ⇒ int → <Keyword, int>

⇒ lexical → only always matches the longest character sequence

(\*) Post increment/decrement in context of equation

⇒ first use value in the eqn.  
then → increment the value

$$\begin{array}{c} a+++b \\ \hline a++ + b \\ 4+3=7 \\ \text{then } a=5 \end{array}$$

(\*) Pre increment/decrement in context of equation

⇒ first increment the value and then use in the equation after completion of the equation

$$\begin{array}{c} a=4 \ b=3 \\ a + \underline{++b} \\ a+4 \leftarrow \\ 4+4=8 \end{array}$$

HW  $a=4$   
 $b=3$

$$\begin{array}{c} a+++\underline{++b} \\ \hline 4+4=8 \end{array}$$

Relational operator → will return either true / false

equal to =

not equal to !=

less than or equal to <=

greater than or equal >=

less than <

greater than >

Logical operator

&&, ||, !

and, or, or not

And, OR → used  
to combine  
2 condition

② && → returns true → all the condition  
want to be true.  
→ returns false → one or more than  
condition is false

③ || → returns → true → one or more the  
one condition will be  
true.  
→ returns false → one or all conditions  
are false

! → returns true → when condition is false

false → when all condition is true

Short circuit → condition anywhere in the  
expression that returns false, then the  
rest of the condition after will not evaluated.

### Bitwise operator

AND      2  
OR      1  
NOT     ~  
left shift    <<  
right shift    >>  
XOR      ^

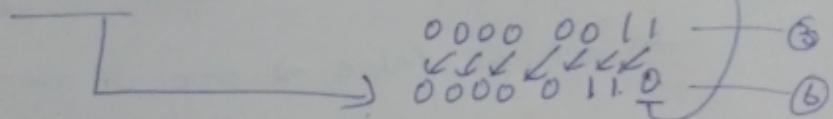
left shift    → 2 operand needed.  
<<

binary operator

① → when bits → shifted left - then trailing pos are filled with zero

ex

Var << 1      Var = 3



② Left shifting is equivalent to multiplication by 2<sup>nd</sup> right operand

Var = 3      → 0000 0011

Var << 1      0000 0110  
→ 3 × 2<sup>1</sup>

Right shift    >>    → binary operator

① bit are shifted right, leading pos are filled with '0'

Var >> 1      0000 0011 ⇒ ③

leading pos → ③ 000 0001 = ①

② Right shift is a division by 2 right operand

Var = 3

$$\text{Var} \gg 1 \rightarrow (3/2)$$

Bitwise-XOR

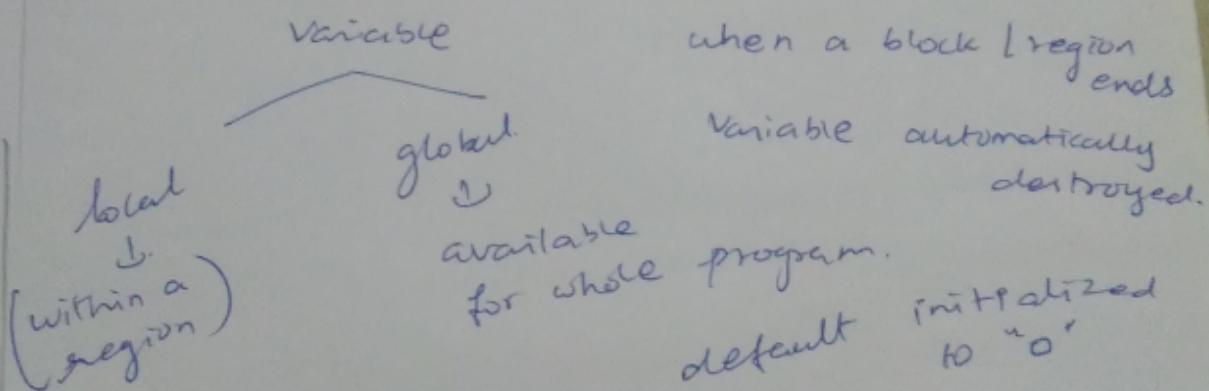
$$\begin{array}{r} 7 \\ 4 \\ 3 \\ \hline \end{array}$$
$$\begin{array}{r} 0111 \\ 0100 \\ 0011 \\ \hline \end{array}$$

$$7 \wedge 4 = 3$$

X-OR		
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

slope = life time  $\rightarrow$  variable is declared, defined & used and

when a block / region ends



Variable modifier  $\Rightarrow$  Auto & extern

Auto modifier auto int Some\_variable\_name

$\hookrightarrow$  it have some garbage value at initial stage

Extern modifier

auto int var -syntax

$\hookrightarrow$  simply means  $\rightarrow$  declaration  
 $\rightarrow$  no memory space

Syntax: extern int a

Modified reg.

→ reg keyword

↳ hints the compiler to  
store a variable in reg  
memory