Chapter 4 :- Operators & Expressions

4.1 : Arithmatic operators

- unary - only one operand — x++, x--, ++x, -y
- binary - requires two operands — x+7, x*y, x^2, x/y

Note :- (modulus) with floating point is big NO NO :

(i) when both integer - o/p is int

(ii) when both floating - o/p is float

(iii) when one int one float - o/p is float

Imp (iv) when +ve % -ve → o/p +ve } output sign will sign of numerator

*** when -ve % +ve → o/p -ve

4.2 : Assignment operators :

x = 20 + x + 9 → assignment expression

x = 20 + x + 7 ; → — ll — statement (with semicolon)

left side == variable while right side can be variable, operator, constant

* compound assignment

imp ***

x = x + 5	y = y / 5	z = z - 20	k *= 5 + x
x += 5	y /= 5	z -= 20	k = k * (5 + x)

4.3 :- Increment / Decrement operators :

(++x) x = x + 1 (x++) } format

(--y) x = x - 1 (y--)

x++ x-- } ✓ valid

constant → 5++ 5-- } ✗ invalid ← only valid with unary operators

non unary operator → (x+y+z)++ Not valid with constants

4.3.1 : Prefix increment / decrement

++x → value of x is incremented first & then used

--x → value of x is demented first & then used

y = ++x ≅ x = x + 1, y = x y = --x ≅ x = x - 1, y = x

4.3.2 : Postfix increment / decrement

x++ → value used & then incremented

x-- → value used & then decremented

y = x++ ≅ y = x, x = x + 1 y = x-- ≅ y = x, x = x - 1

4.4 :- Relational operators

4.5 : Logical operators

4.6 : Conditional Operators :

- Unary operator (? and :) - Test ? result 1 : result 2
- e.g: m = a>b ? x : y

  if a>b == True m = x

  a<b == False m = y

4.7 : Comma operator :

- Used to create compound expression

from x=9 ;

b=2 ;

c=20 ;

z = x+b+c

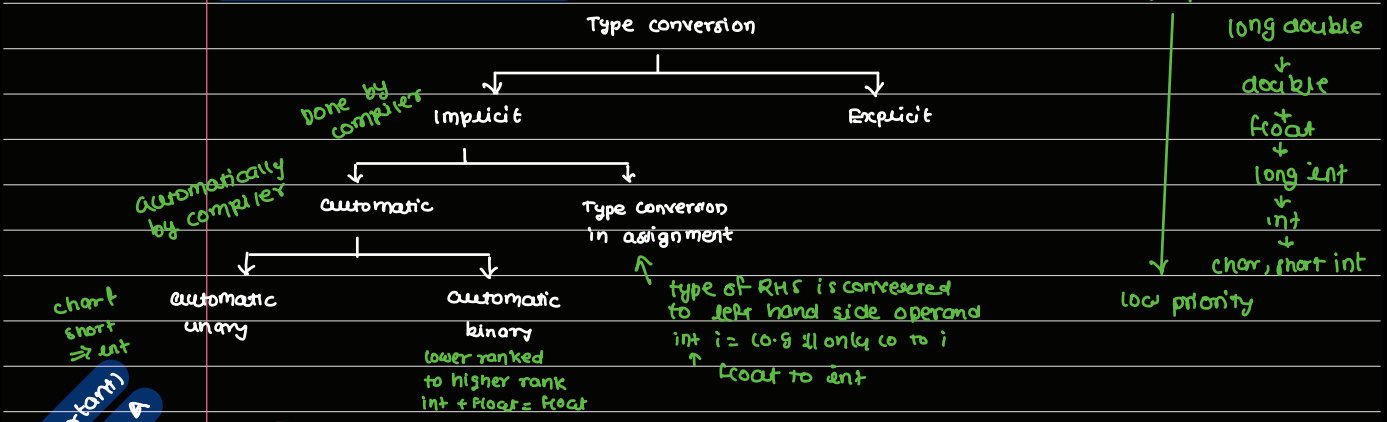
z = (x=9, b=2, c=20, x+b+c);

=> Note ( ) are very imp here

4.8 : Sizeof Operator :

- Unary operator that tells the size in bytes of operant
- e.g: sizeof(x) sizeof(arr) sizeof(double) , sizeof('x')

4.10 : Type Conversion :



Important!!

* Explicit (Type cast)

- Sometimes automatic conversion may not yield actual result e.g

float z, int x=20, y=3 ;

Wrong - z = x/y = 20/3 = 6.0 ← output will be 6.00 & not 6.66

Here we can provide our own truncation called type casting

Correct - z = (float) x/y => converts x to float 20.0/3 => 6.66

Note z = (float)(x/y) => converts (x/y) to float (20/3) => 6.00

- so keep usage of brackets in mind!

(float) while is cast operator

Important AAA

4.11: Precedence & Associativity of Operators :

The below figure is enough to understand this part:

Precedence and Associativity of operators

Operator	Description	Precedence level	Associativity
( ) [ ] → .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ -- ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes	2	*Right to Left*
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
== !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	*Right to Left*
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	*Right to Left*
,	Comma operator	15	Left to Right

Higher to lower precedence table

The table is very important for snippers

4.18: Order of evaluation of operands

- Order of evaluvion of operands of an operation is not defined in C
- z = (++x) - (x--) : output is compiler dependent so avoid such shit.
- imp — Only logical AND, logical OR, conditional operator (?, :) and comma operators (,) have fixed order of evaluation of operands left to right order of execution for above all.