Number System

(2) Binary = {0, 1}
(8) Octal = {0, 1, 2, 3, 4, 5, 6, 7}
(10) Decimal = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
(16) Hexadecimal = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, B, F}

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	B
15	1111	17	F
16	10000	20	10

* Decimal → Binary

$(25)_{10} = (11001)_2$ ✓

$$\begin{array}{r|l}
2 & 25 \\
2 & 12 \to 1 \\
2 & 6 \to 0 \\
2 & 3 \to 0 \\
2 & 1 \to 1 \\
 & 0 \to 1
\end{array} \uparrow$$

* Binary → Decimal

$$\left.\begin{matrix} 1 & 1 & 0 & 0 & 1 \\ 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \end{matrix}\right\} 1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$$

$= 16 + 8 + 0 + 0 + 1$

$= 25$ ✓

Compiler vs Interpreter

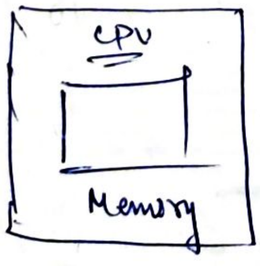
1. Check error (byte code = error free code)
2. Converts into machine code
3. Execution

* C++ → Compiler language.

• Check error → first.exe → RUN → Not job of compiler

(first.cpp) ↳ translation → one time

* Java script → Interpreter based language



• checks error

↳ even if your error is on line 6th,
it will execute your program till line 5.

∴ Every line → translation takes place.

* Hybrid language → Java, C#

$$\begin{tikzcd}
\text{Java, C\#} \arrow[d, "\text{Compiler}"] & \\
\text{byte code} \arrow[d, "\text{JVM}"'] & \\
\text{Machine code} & \text{Interpreter} \arrow[ul, ""]
\end{tikzcd}$$

Machine code
+
Execution

Data Types

$$\begin{tikzcd}[column sep=small]
 & & & \text{Data Types} \arrow[dll] \arrow[d] \arrow[drr] & & & \\
 & \text{Primitive} \arrow[dl] \arrow[d] \arrow[dr] & & \text{User defined} \arrow[d] \arrow[dd] \arrow[ddd] \arrow[dddd] & & \text{Derived} \arrow[d] \arrow[dd] & \\
\text{Integral} \arrow[d] \arrow[dr] & \text{bool} & \text{floating points} \arrow[d] \arrow[dr] & \text{Enum} & & \text{Array} & \\
\text{int} & \text{char} & \text{float} & \text{double} & \text{Structure} & \text{Pointer} & \vdots \\
 & & & & \text{Union} & & \\
 & & & & \text{Class} & &
\end{tikzcd}$$

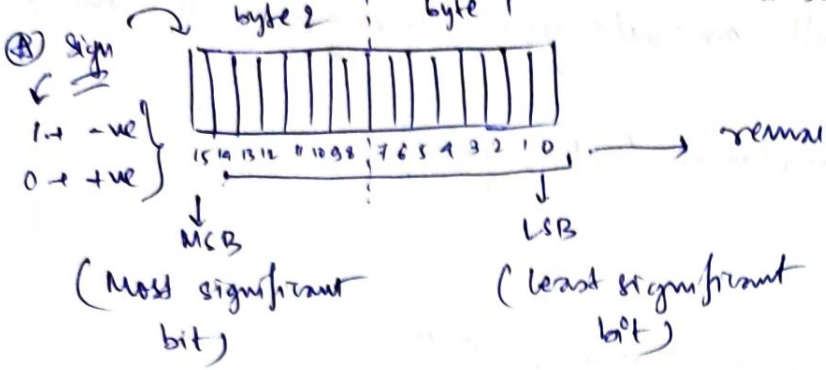
Data type	Size (byte)	Range
int	2 or 4	-32768 to 32767
float	4	-3.4×10^{-38} to 3.4×10^{38}
double	8	-1.7×10^{-308} to 1.7×10^{308}
char	1	-128 to 127
bool	undefined	true/false

• 1 byte = 8 bits

① −32768 to 32767

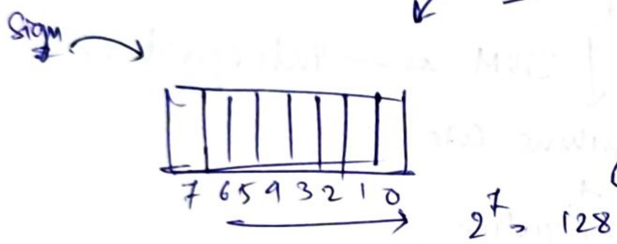
int (2 byte)

−32767 ← −0   0 → 32767



remaining bits = 2^{15}
= 32768

char (1 byte)



$2^7 = 128$ → −128 to 127 (range)

Characters into numbers → how? ⇒ ASCII

Eg →	A − 65	a → 97	0 → 48
	B − 66	b − 98	1 → 49
	⋮	⋮	⋮
	Z − 90	z → 122	9 → 57

* Modifiers
- ↳ unsigned → (+ve) → unsigned int (0 − 65535) 2^{16} ④
  - unsigned char (0 − 255) 2^8 ④
- ↳ long

long int → 4 bytes (if int is 2 byte)
8 bytes (if int is 4 bytes)

long double → 10 bytes

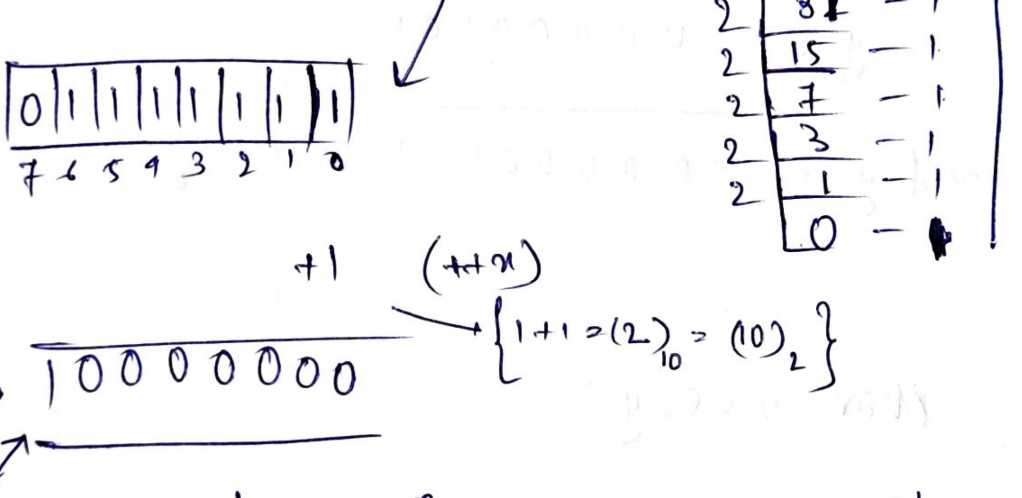
Operator precedence

( )
*, /, %
+, -

Overflow

```
char x = 127.
++x ;
cout << (int) x ;
output → -128
```

range of char
-128 to 127



$2^8 = -128$ → that's how it cycles back

Bitwise operator

bit 1	bit 2	and &	or |	xor ^
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Eg int x = 11, y = 5;

```
    x =   00001011
    y =   00000101
  ----------------
x & y =   00000001   = (1)
```
$= (1)_{10}$

char x = 5, y;

```
  x →      00000101
y = ~x →   11111010
```

sign → -ve → 2's compliment.

```
  11111010
  00000101
        +1
  --------
  00000110
```

$\boxed{110} \to 6$

So, not of 5 is 6

* Left shift

int x = 5, y;

x = 00000101

y = x << 1   00001010 → 10

So, 5<<1 = 10

⊛ x<<i

Ans → $\boxed{x * 2^i}$

* Right Shift

x>>i

Ans → $\boxed{\frac{x}{2^i}}$

⊛ Enum (User defined data type)

Dept

CS	- 0
ECE	- 1
IT	- 2
EE	- 3
ME	- 4

→ enum dept {$\overset{0}{\text{CS}}$, $\overset{1}{\text{ECE}}$, $\overset{2}{\text{IT}}$, $\overset{3}{\text{EE}}$, $\overset{4}{\text{ME}}$}

or enum dept {CS = 1, $\overset{2}{\text{ECE}}$, $\overset{3}{\text{IT}}$, $\overset{4}{\text{EE}}$, $\overset{5}{\text{ME}}$}

↳ if i want ~~the~~ numbering to start w/ 1

⊛ Typedef

```
int main()
{  int m1,m2,m3,r1,r2,r3;
      :
}
```

→

```
typedef int marks;
typedef int roll;
int main()
{
     marks m1,m2,m3;
     roll  r1,r2,r3;
```