Conditionals ⇒ true false

Statement ⇒ (x >= 2)

==
>
<
!=
>=
<=

true : x = 3, 4, 5, 2, - -
false : x 0 -1, 1, - -

Conditional ⇒ boolean value ⇒ value

bool flag = ~~true~~ false;

if (false) {
~~execute this~~
}

```
if (0) {
    ignored
}
```

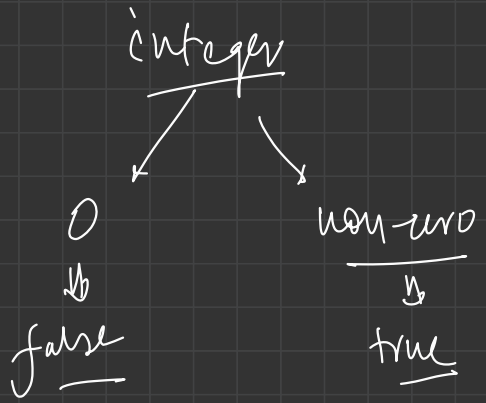
false

0 → is false value

```
x ≠ 0
if (x) {
    ---- execute
}
```

true

any non-zero integer ⇒ true



Swapping

trying to get the address of variable

we need to pass (buckets) as arguments.

a chunk of memory at some location

{ it will have some address }

int → 4 byte
bool → 1 byte
float → 4 byte
char → 1 byte

Byte is the smallest unit of memory we use.

1 Byte = 8 bits

→ every byte should have some address ??

8GB ⇒ $8 \times 1024 \times MB \Rightarrow 8 \times 1024 \times 1024\ KB \Rightarrow 8 \times 1024 \times 1024 \times 1024 \times$ Bytes

$2^3 \times 2^{10} \times 2^{10} \times 2^{10} = 2^{33}$ Bytes of memory

2^{10} addresses → every address would be of 10 digits if we use decimal representation

add. 2451098763

→ binary representation ⇒ 33 digits (bits)

01101100 ... 00 ... — 33 bits

01100...110 33 bits long

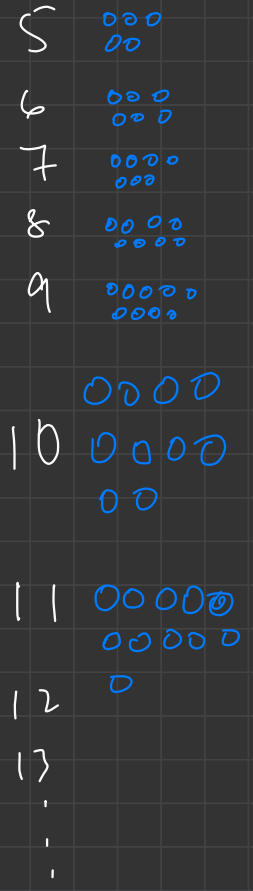
Number system: to represent memory addresses

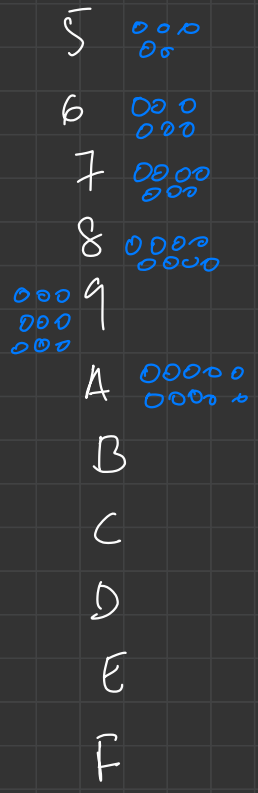
Hexadecimal (base-16)

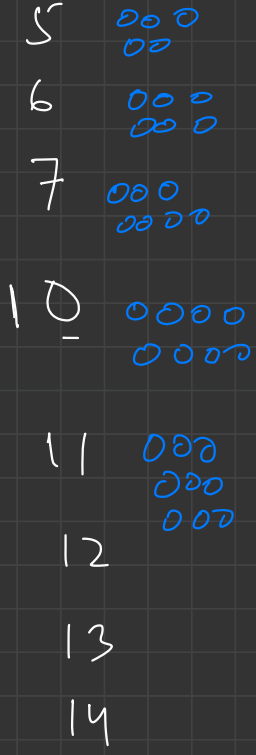
if has 16 symbols

2 Binary	10 Decimal	16 HexaDecimal	8 Octal
0	0	0	0
1	1	1	1
	2	2	2
	3	3	3
	4	4	4

Decimal







Decimal

0 1 2 3 4 5 6 7 8 9 10 11 12 13

+1 +1 +1

14 ..... 19 20 21 .. 99 ... 100 101 ... 999 ... 1000

Octal

0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17

+1

~~18~~ 20 21 22 23 24 .. 27 30 31 .. 77 .. 100 ... 777

1000 ...

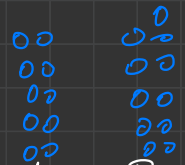
Binary

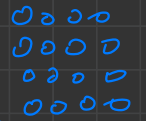
99 → 100

109 110

0 1 10 11 100 101 110 ... 111 1000 ...

Hexadecimal



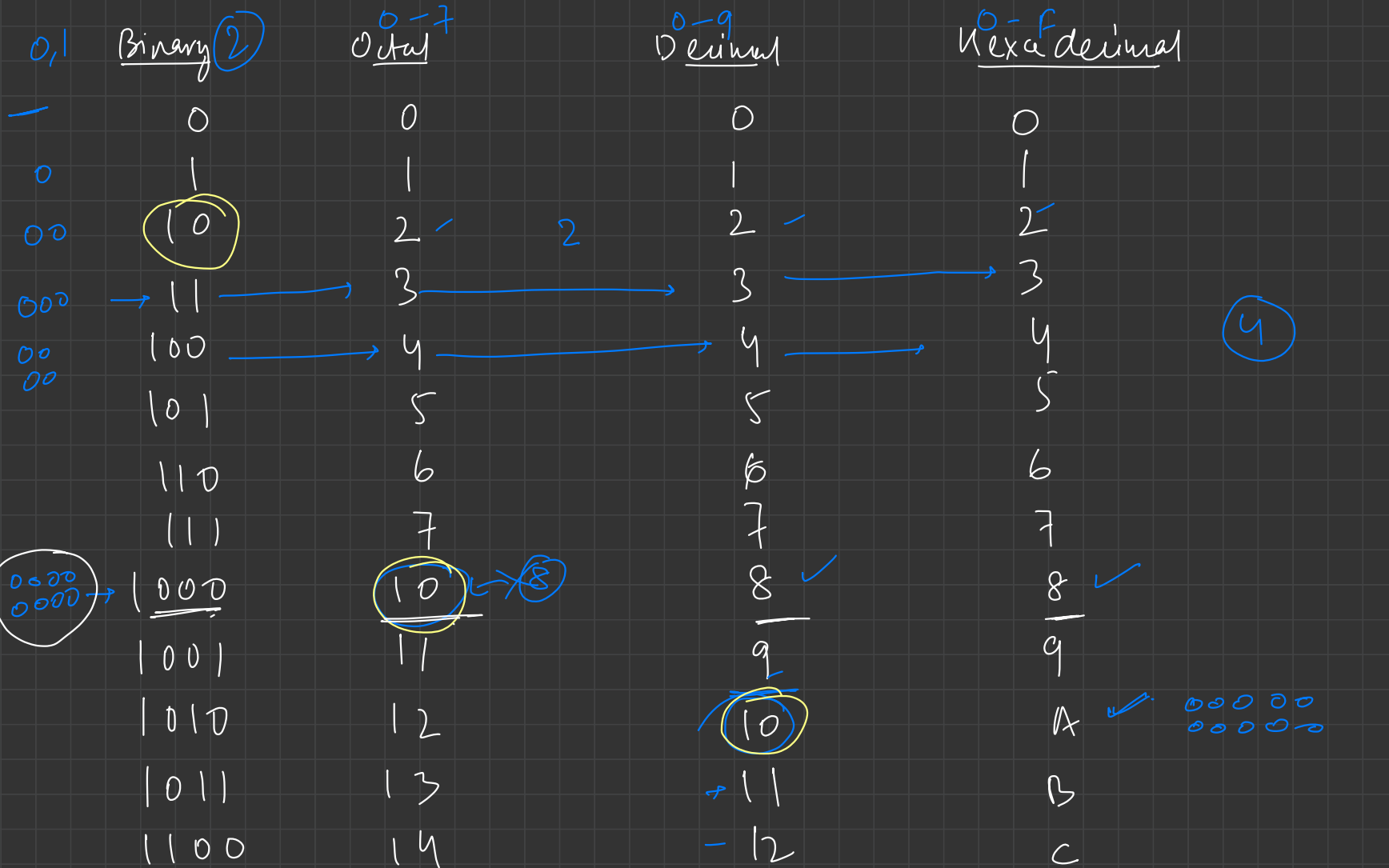


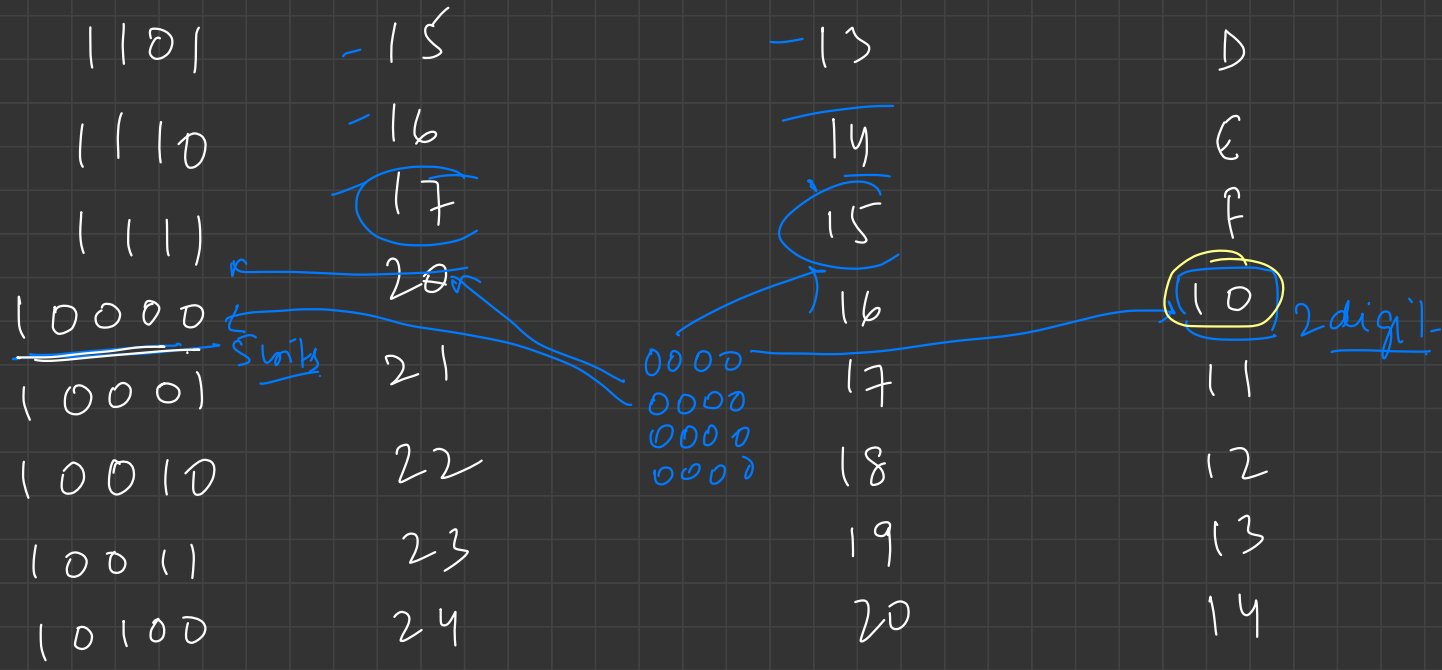
0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11

10 11 12 13 14 15 17

12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20

21 22 .... FF 100 101 102


0,1
Binary (2)
Octal 0-7
Decimal 0-9
Hexadecimal 0-f
0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
0
1
2
3
4
5
6
7
10
11
12
13
14
2
0
1
2
3
4
5
6
7
8
9
10
11
12
0
1
2
3
4
5
6
7
8
9
A
B
C
4
8



0 1

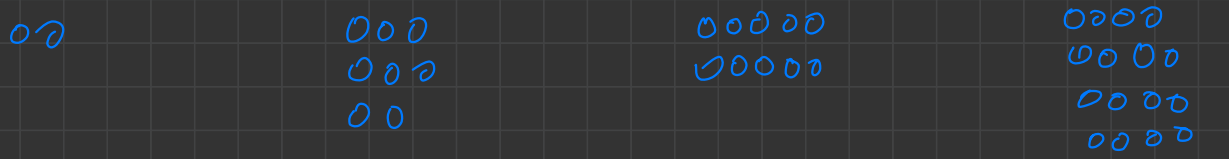
0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9 A B C D E F

When the symbols end, make it zero, and go the next symbol at left.

$(10)_2$ $(10)_8$ $(10)_{10}$ $(10)_{16}$



We need more digits to represent the quantity when using a lower number system (having less symbols)

more the symbols available $\Rightarrow$ less digits needed to represent the quantity/value

Decimal

10^4 10^3 10^2 10^1 10^0

Binary

2^4 2^3 2^2 2^1 2^0

Octal

8^4 8^3 8^2 8^1 8^0

Hexadecimal

16^4 16^3 16^2 16^1 16^0

$(12345)_{16} \Rightarrow 5 \times 16^0 + 4 \times 16^1 + 3 \times 16^2 + 2 \times 16^3 + 1 \times 16^4$

$\Downarrow$

$(\quad\quad)_{10}$

$(12345)_8$

$(0\ 1001)_2$

$(12345)_{10} = 5 \times 10^0 + 4 \times 10^1 + 3 \times 10^2 + \cdots$

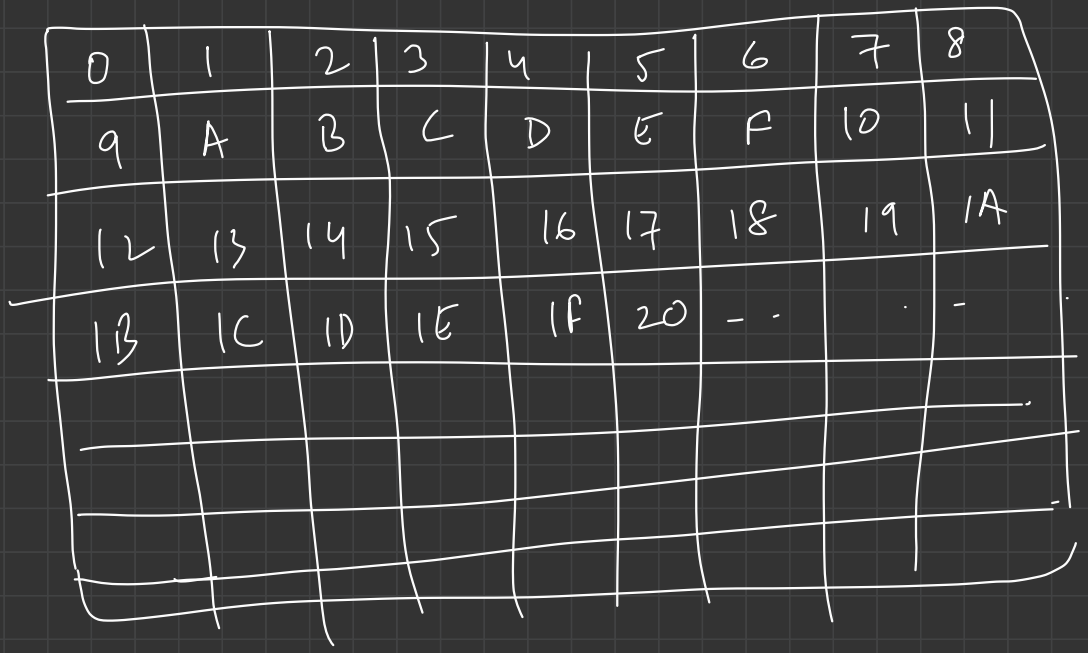
Addresses too many

we use hexadecimal number systems to represent memory addresses

$(\quad)_{16}$ ⇒ 0x54123 $= (54123)_{16}$

Hexa rep.

A B C D E F.



Memory addresses are represented in Hexadecimal

How to get the address of a variable?!

⇓

address-of operator. (&)

Swap . pass the address as arg.

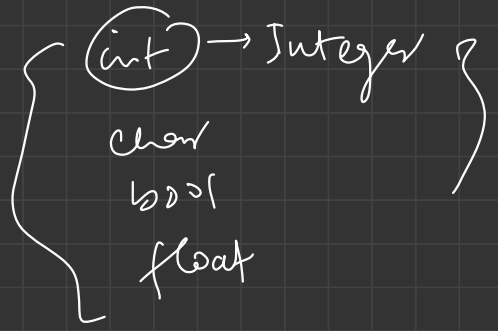
① get the address ② then pass it as an argument / parameter

③ get the value at the address.

pointer pointer

returntype name (type name, type name){

}



swap(&a, &b);

Pointers. ⟸ we had no data type to store address

↓

is a variable, that holds some address (another variable)

pass the address instead of value

Call by reference

- address op
- pointers
- dereference

When we declare a pointer, it initially stores/points to a random address/value, but we should not access it.

`int *p;` ⟶ p is a pointer pointing to a random address.

`int *p=NULL;`

`=0`

`=nullptr;`

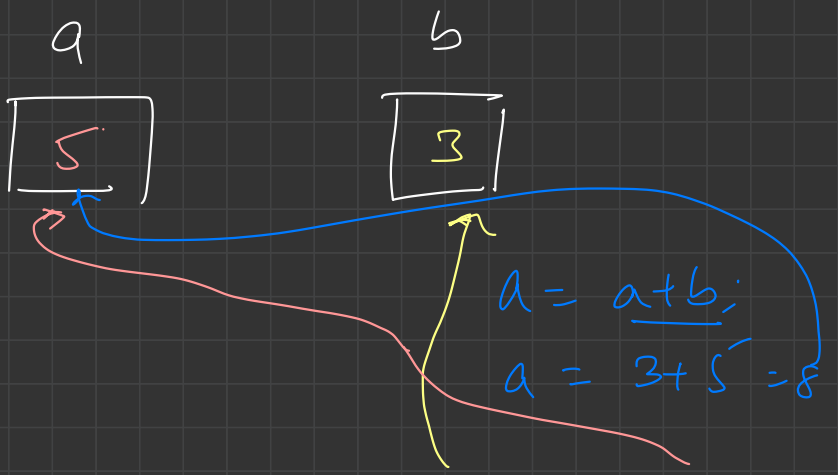
`*p` → is wrong illegal.

NULL 0 nullptr same thing

```
int * ptr = &a;
```

Swapping without third variable

```
a = a+b;
b = a-b;
a = a-b;
```



a = a-b
= 8-3 = 5

b = a - b;
= 8 - 5 = 3
b = 3

a = 5

a
7

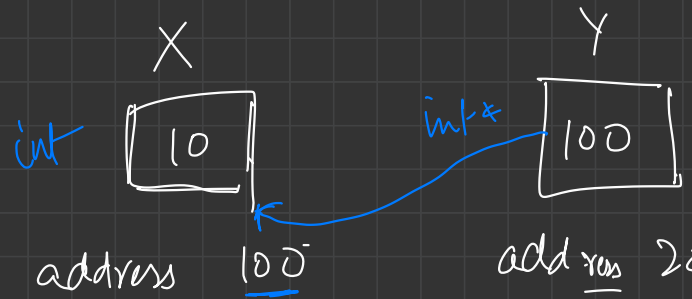
b
4

a = a x b;
= 4 x 7 = 28
a = 28

a = a/b
= 28/4
a = 7

b = a/b;
= 28/7 = 4
b = 4

division by 0 is not allowed.




```
int *y = &x;
```

x → data type : int

y → data type : pointer (int *)

---

① x → 10 ✓

② &x → 100 ✓

③ y → 100 ✓

④ &y → 200 ✓
100

⑤ *(&x) → 10, 100, ~~error~~

⑥ &(*y) → 100, 10, 200, V(1000)

⑦ x+1 → 11

⑧ (&x)+1 → 101, ~~error~~, 11

⑨ y+1 → 100, 201, 101

⑩ (*y)+1 → 11, 11, 101, ~~error~~

(11) *(&y) ⟶ 10, ~~100~~, 101, 100