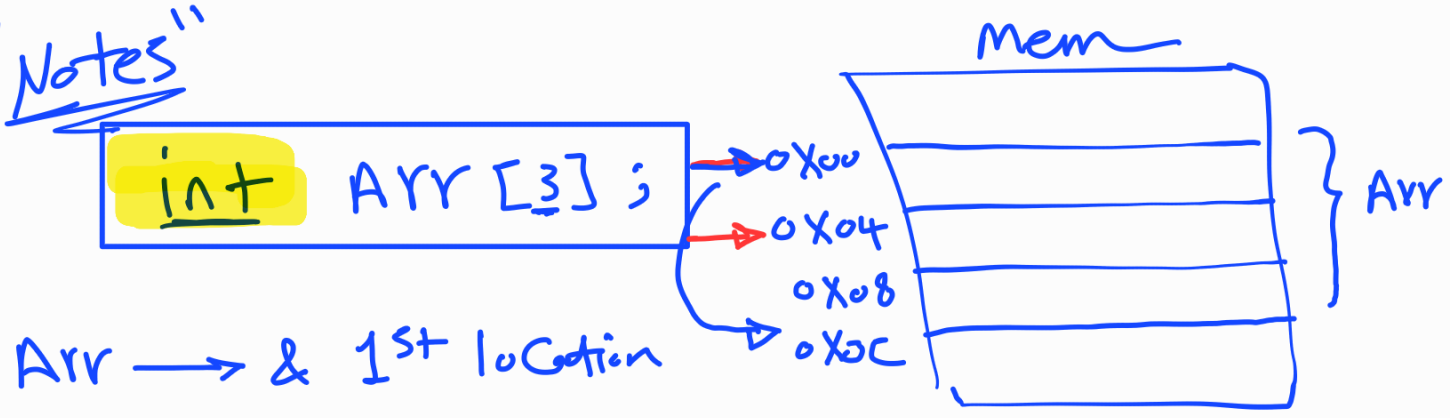
Array - P2

"Notes"



Arr → & 1st location

Printf ("%p \n" , Arr); → 0X00

&Arr → 0X00

Arr+1 → 0X00+4 = 0X04 (int)

↳ step = ? → element

&Arr+1

↳ step = ? → Array

= 3 * 4 = 12 b

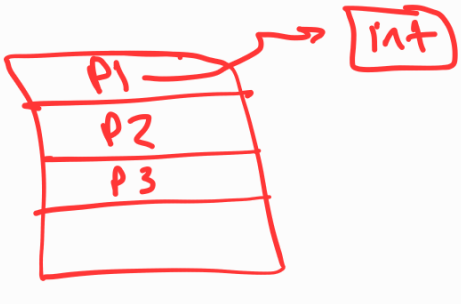
Conc

arr → Name ≡ Const Pointer to [element data type]

&arr → Const Pointer to [Array] → step

✱ Array of Pointers :-

int *P [3];




```
P[0] = &X;
P[1] = &Y;
P[2] = &Z;
```

why?

→ Array of Pointers to funcs

→ Calling & Call back func

Passing Arrays as func Arguments.

two methods

① Formal Parameters as an unsized array

② ~ ~ a Pointer

eg

```
Void func ( int Arr[] )
{
}
```

eg

```
Void func (int *Arr)
{
}
```

hint

↳ Pass the Array Size

↳ (sizeof(Arr) / sizeof Arr[0])

Ex → Print Array using func

↓ for loop

↓ & 1's + loca

↓ Size

⇒ Multi-Dimensional Arrays

Arrays → # dimen

2 → 2D Array (Matrix

3 → 3D Array (Cube

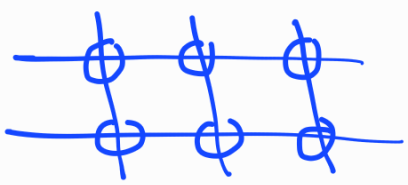
4 ------ #

(*) General format Multi-dim Array :-

Type Name [size1] [size2] ---- [sizeN];

Ex 2D → int Arr[2][3];

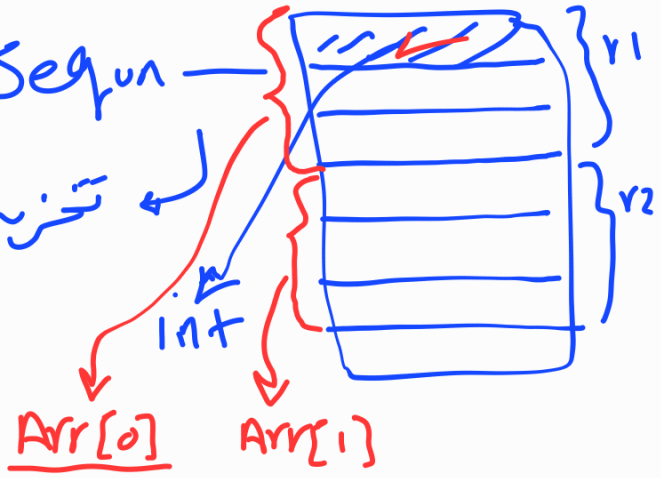
rows Columns



But ⟹ in memory ⟹ Sequn

تخزين صف ورا بعض

eg int Arr[2][3]



⊛ Initialization of 2D Array :-

r0 r1

int arr[2][3] = { , , , , , , };

for Readability

Comma

int Arr[2][3] = { { , , , }, { , , , } };

Notes

✗ int arr[ ][ ] = { , , , , , , };

✗ int arr[2][ ] = { , , , , . . . };

✓ int arr[ ][3] = { , , , , , , };

→ init - while declare

⊛ Accessing 2D Array

int var = Arr[2][5];

→ Subscript operator

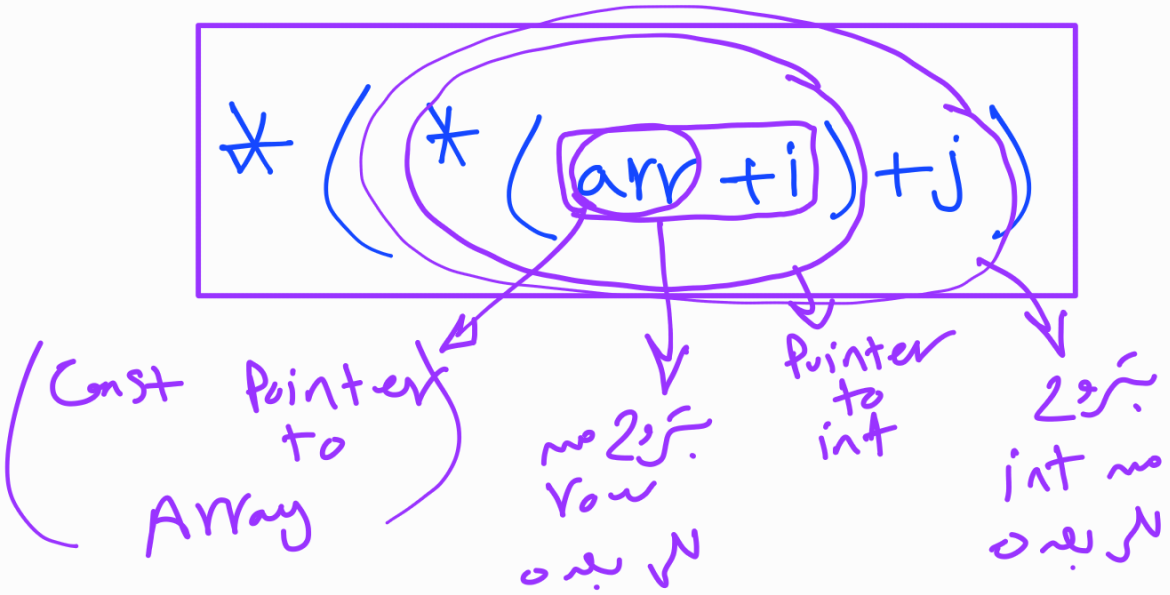
or

arr[i][j]

((arr+i)+j)

Name of 2D Array → Pointer to Array

یعنی arr+1



arr → 2D

arr → step = Array

&arr → step = 2D array

Variable Length Array:

Size → Const integ → ANSI-C (C89)

No → Var / run time

• VLA → C99 ⇒ flexible array