CS & IT ENGINEERING

Programming in C

Arrays and Pointers 03

Lec- 03

By- Pankaj Sharma sir

Arrays-III

BA ⇒ 1000
int ⇒ 4 bytes

```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
printf("%u", a);
printf("%u", &a);
printf("%u", a[0]);
printf("%u", &a[0][0]);
printf("%u", *a);
printf("%u", **a);
printf("%u", a+1);
printf("%u", *a+1);
printf("%u", a[0]+1);
```

Same

&a[0] 1000

2D array

&a 1000

&a[0][0] 1000

1000 1000

a[0][0]

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

*a+1 ⇒ &a[0][0]+1

sizeof a[0][0] ⇒ 4

*a+1 ⇒ &a[0][0] + 1×4 ⇒ 1004

*a ⇒ *&a[0]

⇒ a[0]

⇒ &a[0][0]

a[0] +1 ⇒ &a[0][0]+1

1000

1

1016

1004

1004

← a[0] → ← a[1] → ← a[2] →

a

**a

*&a[0][0]

⇒ a[0][0]

a+1 ⇒ &a[0]+1

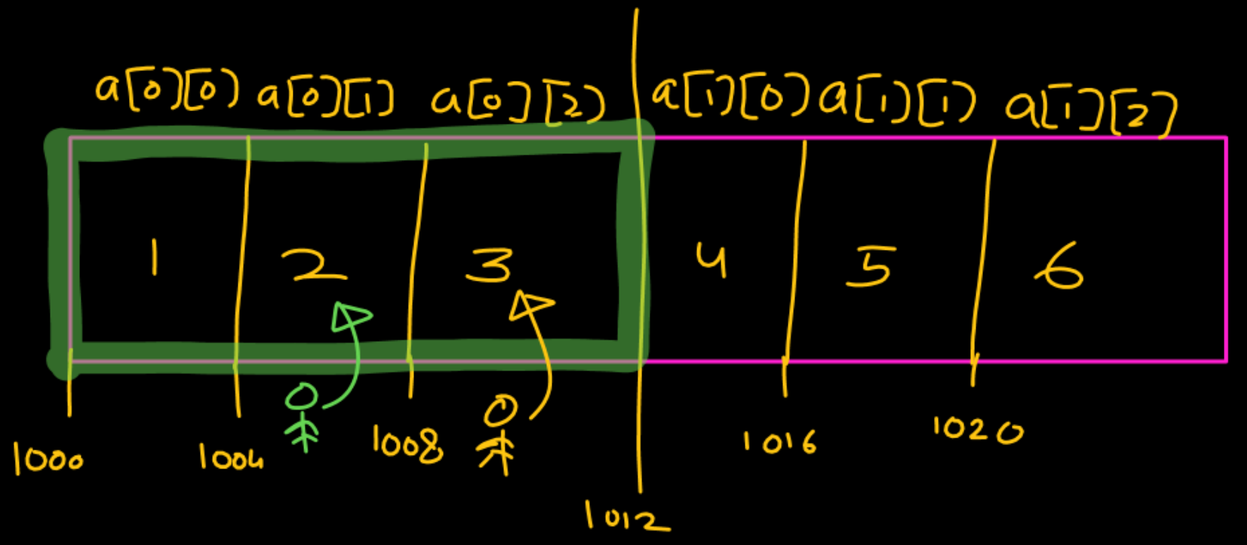
sizeof a[0] ⇒ 16 byte

a+1 ⇒ &a[0] + 1×16

⇒ 1000 + 16

= 1016

```
int a[2][3] = {1,2,3,4,5,6};
```



(i) $a[0] \Rightarrow \&a[0][0]$

sizeof a[0][0] = 4

$a[0]+1 \Rightarrow \&a[0][0] + 1$

$\Rightarrow \&a[0][0] + 1 \times 4 = 1004$

$a[0]+1 \Rightarrow$ Memory location 1004 $\Rightarrow \&a[0][1]$

$*(a[0]+1) \Rightarrow$ value at (memory location 1004) $\Rightarrow *\&a[0][1]$

$*(a[0]+1) = 2 = a[0][1]$

(ii) $a[0]+2 \Rightarrow \&a[0][0] + 2$

$\Rightarrow \&a[0][0] + 2 \times 4$

$= 1008$

$(a[0]+2) \Rightarrow$ Memory location 1008 $\Rightarrow \&a[0][2]$

$*(a[0]+2) =$ value at (memory location 1008) $\Rightarrow *\&a[0][2]$

$*(a[0]+2) = 3 = a[0][2]$

$*(a[0]+1) \Rightarrow a[0][1]$

$*(a[0]+2) \Rightarrow a[0][2]$

$*(a[1]+1) = a[1][1]$

$*(a[1]+2) = a[1][2]$

$*((a[0])+j) \Rightarrow a[0][j]$

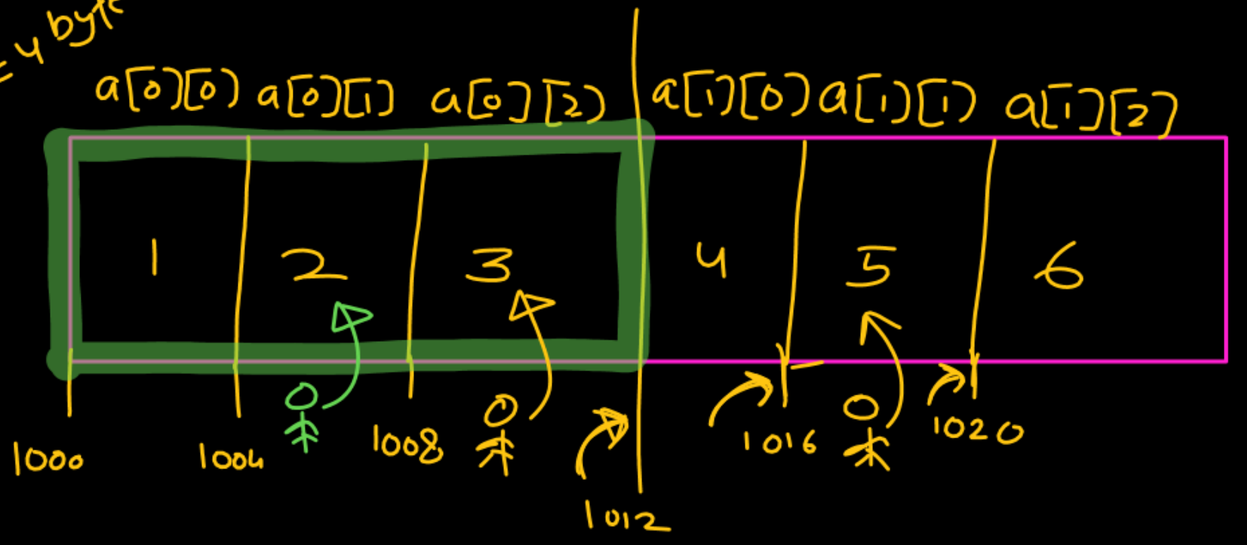
$*(a[1]+j) = a[1][j]$

$$*(a[i]+j) = a[i][j]$$

$$*(a[i]+j) = a[i][j] = *(*(a+i)+j)$$

int a[2][3] = {1,2,3,4,5,6};

size of a[1][0] = 4 byte



(iii) a[1] ⇒ &a[1][0]

a[1]+1 ⇒ &a[1][0] + 1

⇒ &a[1][0] + 1×4

= 1012 + 4 ⇒ 1016

a[1]+1 = Memory location 1016 = &a[1][1]

(a[1]+1) = value at (Memory location 1016) = ~~~~~~&~~a[1][1]

*(a[1]+1) = 5 = a[1][1]

(iv) a[1]+2 ⇒ &a[1][2]

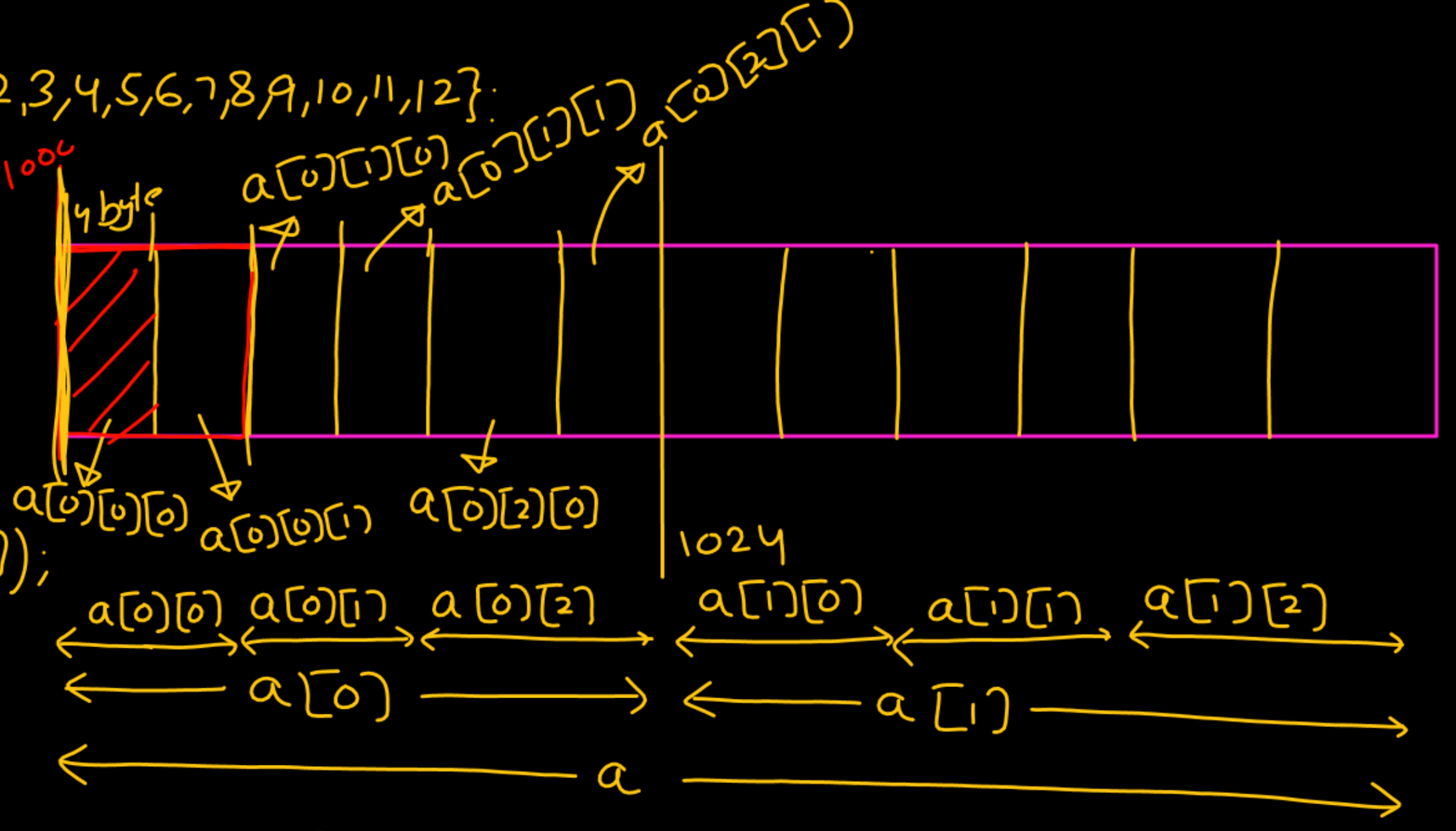
(a[1]+2) ⇒ ~~~~~~&~~a[1][2]

*(a[1]+2) = a[1][2]

BA
1000
4 byte

```
int a[2][3][2] = {1,2,3,4,5,6,7,8,9,10,11,12};

printf("%u", a);
printf("%u", a[0]);
printf("%u", a[0][0]);
printf("%u", a[0][0][0]);
printf("%u", &a);
printf("%u", a+1);
printf("%u", a[0]+1);
printf("%u", a[0][0]+1);
printf("%u", &a+1);
printf("%u", a[0][0][0]+1);
```



$\Rightarrow$ &a[0][0] + 1

$\Rightarrow$ &a[0][1]

BA
1000
4 byte

```
int a[2][3][2] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

```
printf("%u", a);          &a[0]          1000
printf("%u", a[0]);       &a[0][0]       1000
printf("%u", a[0][0]);    &a[0][0][0]    1000
printf("%u", a[0][0][0]); elem           1
printf("%u", &a);  → 1000
printf("%u", a+1);        &a[0] + 1 x 24 = 1024
printf("%u", a[0]+1);     &a[0][0] + 1 x 8 = 1008
printf("%u", a[0][0]+1);  &a[0][0][0] + 1 x 4 = 1004
printf("%u", (&a)+1);  ⇒  &a + 1 x 48 = 1048
printf("%u", a[0][0][0]+1);   1+1 = 2 element
```

$a+1 \Rightarrow \&a[0]+1$

$\Rightarrow \&a[1]$

$\&a + 1$ → element / object

int a[3][2][3] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};

(i) a
(ii) &a
(iii) a[0]
(iv) a[0][0]
} 1000

(v) a[0][0][0]   1

vi) a+1 ⇒ &a[0]+1 ⇒ &a[0]+1×24 = 1024
vii) &a+1 ⇒ &a+1 ⇒ &a+1×72 ⇒ 1072
viii) a[0]+1 ⇒ &a[0][0]+1×12 ⇒ 1012
ix) a[0][0]+1 = &a[0][0][0]+1×4 = 1004
x) a[0][0][0]+1 ⇒ 2

~~value~~
(i) address (ii) ।
ये रखा address है।

xi) *a   *&a[0] → a[0] ⇒ &a[0][0]
1000 xii) **a ⇒ *&a[0][0] = a[0][0] = &a[0][0][0]
1 xiii) ***a   *&a[0][0][0] ⇒ 1
xiv) *a+1   &a[0][0]+1×12 = 1012
(xv) **a+1   &a[0][0][0]+1×4 = 1004

Declaration & Initialization

1) `int a[];` Invalid

2) `int a[] = {10,20,30};` ✓ valid

If only declaration is there without initialization

⟹ It is mandatory to provide the size of each dimension of array.

`int a[][];` Invalid

`int a[][10];` Invalid

`int a[2][3];` valid

(ii) In case, we are initializing the array, there is flexibility that u can omit the size of 1st dimension.

No other dim is having such flexibility.

(i) `int a[];` Invalid

(ii) `int a[2];` ✓

(iii) `int a[] = {10,20};` ✓

(iv) `int a[2] = {10,20};` ✓

(v) `int a[2] = {1};` ✓

(vi) `int a[2] = {10,20,30,40};` ✓

Declaration

(i) `int a[][];` ✗

(ii) `int a[2][];` ✗

(iii) `int a[][3];` ✗

(iv) `int a[2][3];` ✓

(v) `int a[][] = {1,2,3,4,5,6};` ✗

(vi) `int a[][3] = {1,2,3,4,5,6};` ✓

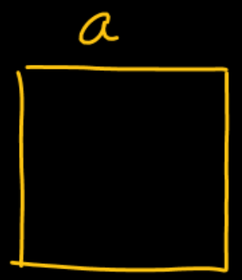
(vii) `int a[2][] = {1,2,3,4,5,6};` ✗

(viii) `int a[2][3] = {1,2,3,4,5,6};` ✓

Pointer

Special variable that are used to store address of other variables.

int a ;



int *p ; *p ⇒ integer

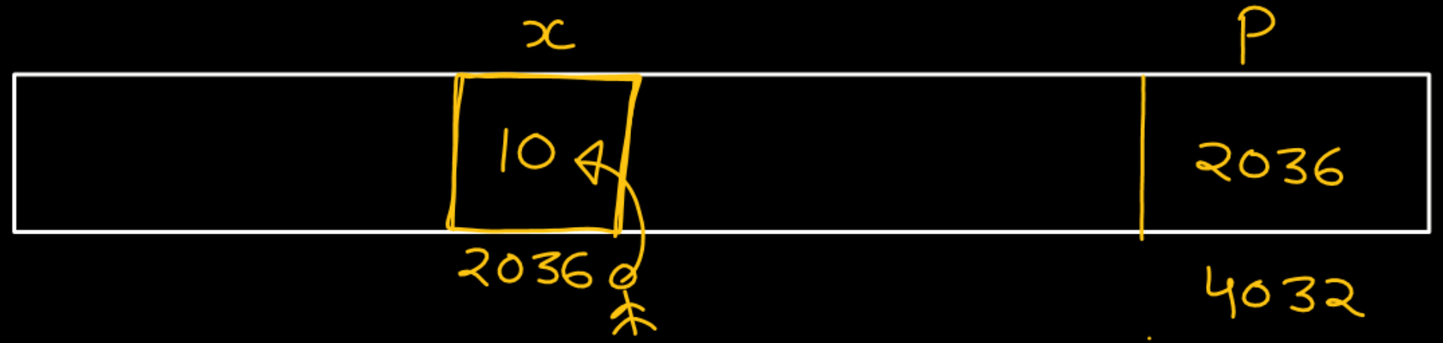
int *p ;

int *p ;

p is a pointer to integer.
p can hold address of same integer variable.

Special
P ⇒ variable ✓

```
int x = 10;
int *p;
p = &x;
```




```
printf("%u", x);   10
printf("%u", P);   2036
printf("%u", *p);  10
```

P = 2036 (Memory location)

*P ⇒ value at (Memory location 2036)

*P = 10

10:00 PM


THANK YOU GW
SOLDIERS !