CS & IT ENGINEERING

C Programming

Arrays and Pointers

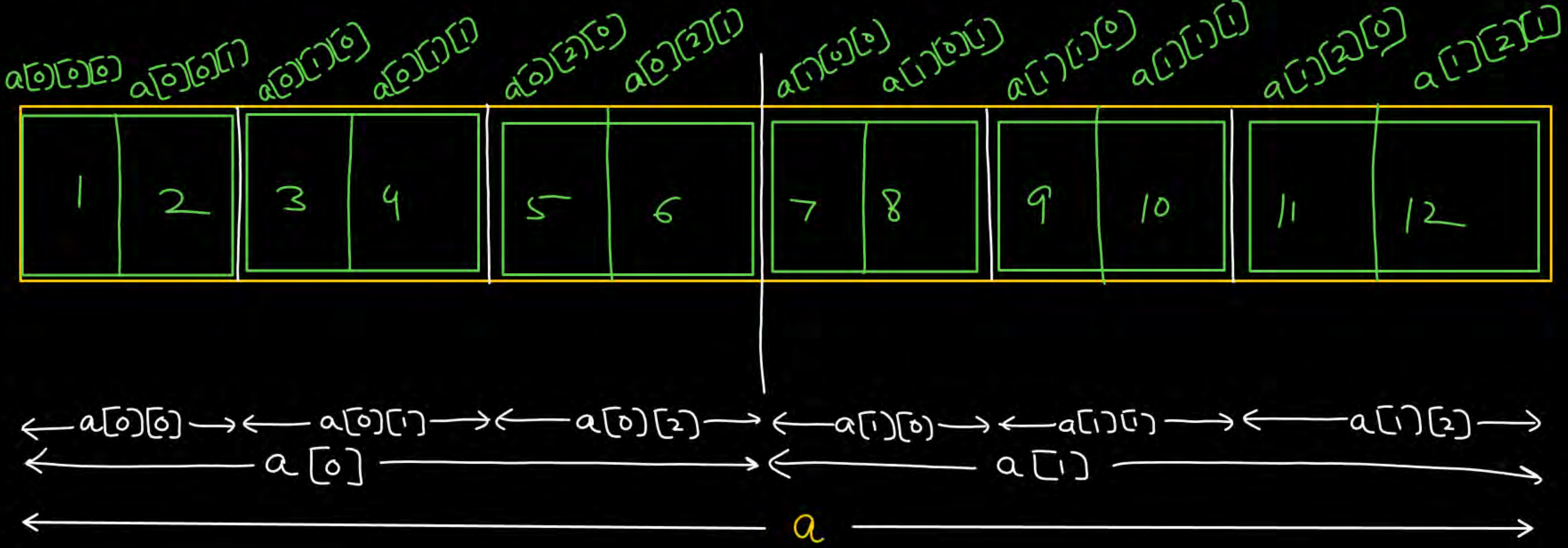
Lec - 04

By- Pankaj Sharma Sir

TOPICS TO BE COVERED

- Arrays and Pointers (Part- 04)

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



a : array a[0], a[1]

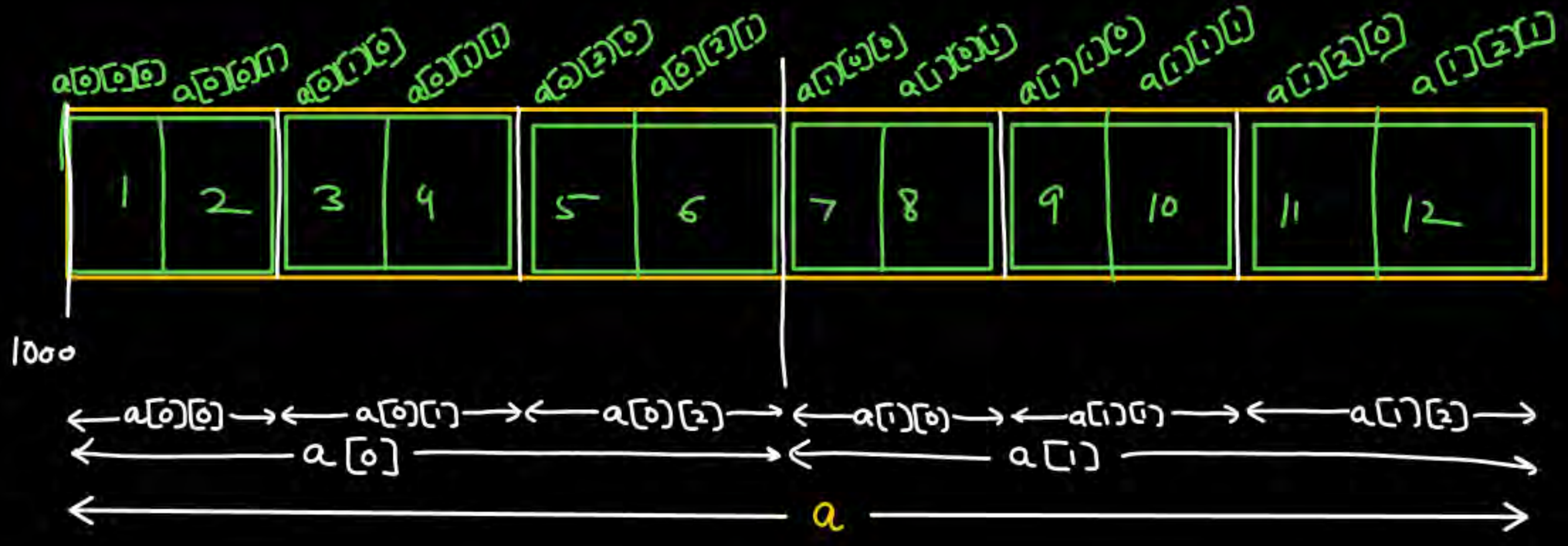
a[0] : array a[0][0], a[0][1], a[0][2]

a[0][0] : array a[0][0][0], a[0][0][1]

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

```
pf("%d", a);
pf("%d",a[0]);
pf("%d",a[0][0]);
pf("%d", &a);
pf("%d",a[0][0][0]); 1
```

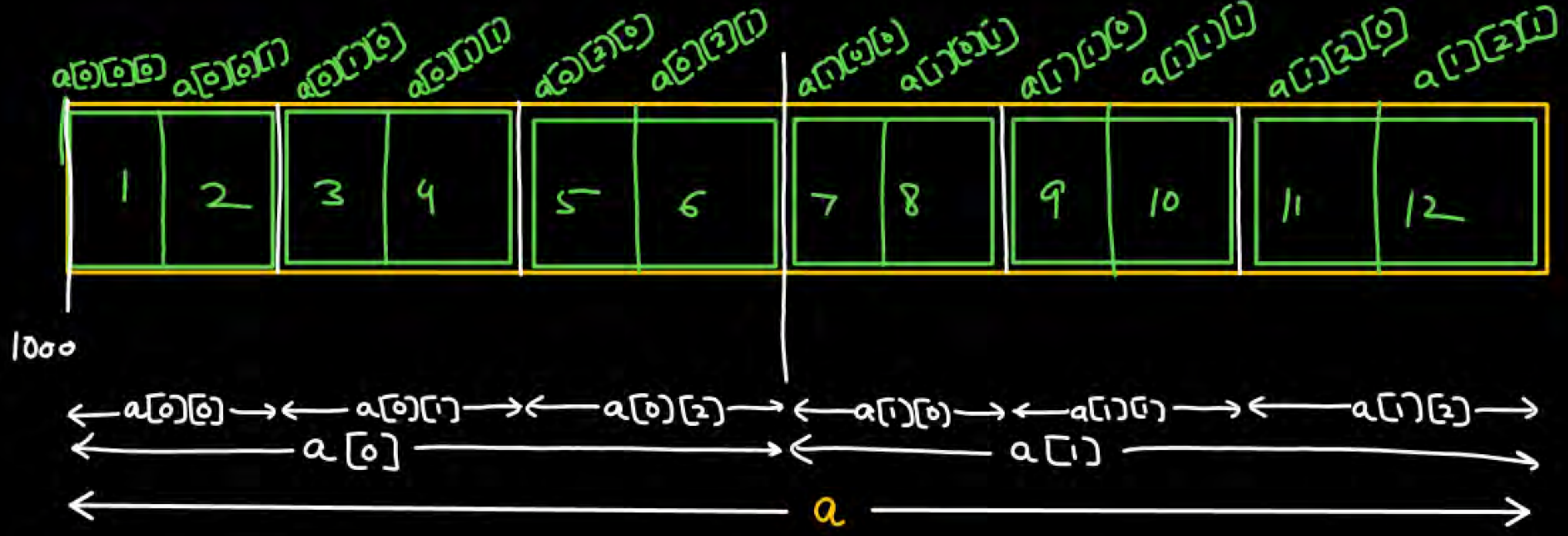
$\left.\begin{array}{l}\texttt{pf("\%d", a);}\\ \texttt{pf("\%d",a[0]);}\\ \texttt{pf("\%d",a[0][0]);}\\ \texttt{pf("\%d", \&a);}\end{array}\right\}$ 1000



Size of int = 4 byte

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

```
pf("%d", a+1);
pf("%d", a[0]+1);
pf("%d", a[0][0]+1);
pf("%d", &a+1);
```



(i) $a+1 = \underbrace{\&a[0]}_{24\text{ byte}}+1$

$= \&a[0]+1\times 24$

$= 1024$

(ii) $a[0]+1 = \underbrace{\&a[0][0]}_{8\text{ byte}}+1$

$= \&a[0][0]+1\times 8$

$= 1008$

iii) $a[0][0]+1 = \underbrace{\&a[0][0][0]}_{4\text{ byte}}+1$

$= 1000+1\times 4$

$= 1004$

iv) $\&a+1 = \&a+1\times 48$

$= 1000+48$

$= 1048$

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

{ st → 1000
int → 4byte
5 min }

$$\left.\begin{array}{ll} \text{(i)} & a \\ \text{(ii)} & a[0] \\ \text{(iii)} & a[0][0] \\ \text{(iv)} & \&a \end{array}\right\} 1000$$

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

vi) a+1

vii) a[0]+1

viii) a[0][0]+1

ix) &a+1

x) a[0][0][0]+1

xi) *a

xii) **a

xiii) ***a

xiv) *a+1

xv) **a+1

xvi) ***a+1

xvii) *a[0]+1

xviii) **a[0]+1

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

{ st → 1000
int → 4byte
5 min }

(i) a

(ii) a[0]

(iii) a[0][0]

(iv) &a

} 1000

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

vi) a + 1

vii) a[0] + 1

viii) a[0][0] + 1

ix) &a + 1

x) a[0][0][0] + 1

xi) *a

xii) **a

xiii) ***a

xiv) *a + 1

xv) **a + 1

xvi) ***a + 1

xvii) *a[0] + 1

xviii) **a[0] + 1

vi) $a+1 \Rightarrow \underbrace{\&a[0]}_{24\text{byte}} + 1 = \&a[0] + 1 \times 24$

$= 1000 + 24 = 1024$

vii) $a[0]+1 \Rightarrow \underbrace{\&a[0][0]}_{12\text{byte}} + 1 = \&a[0][0] + 1 \times 12$

$= 1012$

viii) $a[0][0]+1 \Rightarrow \underbrace{\&a[0][0][0]}_{4\text{bytes}} + 1 = \&a[0][0][0] + 1 \times 4$

$= 1004$

ix) $\underbrace{\&a}_{72\text{bytes}} + 1 \Rightarrow \&a + 1 \times 72 = 1072$

x) $a[0][0][0] + 1 = 1 + 1 = 2$

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

{ st → 1000
int → 4byte
5 min }

xi) $*a \rightarrow \cancel{*}\cancel{\&}a[0] \Rightarrow a[0] \Rightarrow \&a[0][0] \Rightarrow 1000$

xii) $**a \rightarrow *(\&a[0][0]) \Rightarrow \cancel{*}\cancel{\&}a[0][0] \Rightarrow a[0][0] \Rightarrow \&a[0][0][0] \Rightarrow 1000$

xiii) $***a \rightarrow *(**a) = *(\&a[0][0][0]) \Rightarrow \cancel{*}\cancel{\&}a[0][0][0] = a[0][0][0] = 1$

xiv) $*a+1 \rightarrow \underbrace{\&a[0][0]}_{12\text{ byte}} + 1 \Rightarrow \&a[0][0] + 1\times 12 = 1012$

xv) $**a+1 \rightarrow \underbrace{\&a[0][0][0]}_{4\text{bytes}}+1 = \&a[0][0][0]+1\times 4 = 1004$

xvi) $\boxed{***a}+1 \rightarrow 1+1 = \boxed{2}$ ✓

xvii) $*a[0]+1 \rightarrow *a[0]+1 \Rightarrow *(\&a[0][0])+1 = \cancel{*}\cancel{\&}a[0][0]+1 = a[0][0]+1$

$= \underbrace{\&a[0][0][0]}_{4\text{byte}}+1$

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

$= \boxed{1004}$

xviii) $**a[0]+1$

$\downarrow$

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

$= *(\&a[0][0][0])+1$

$= \cancel{*}\cancel{\&}a[0][0][0]+1 = a[0][0][0]+1$

$= 1+1 = 2$

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

{ st → 1000
int → 4byte
5 min }

$**a[0] + 1$

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

(ii) $*a[0] \Rightarrow \cancel{*}\cancel{\&}a[0][0]$

$*a[0] = a[0][0]$

(iii) $a[0][0]$ → add ✓
→ ele ✗

$*a[0] = a[0][0] = \&a[0][0][0]$

$*a[0] = \&a[0][0][0]$

$**a[0] = *\&a[0][0][0]$

$**a[0] = \cancel{*}\cancel{\&}a[0][0][0]$

$= a[0][0][0]$

$**a[0] = 1$

$**a[0]+1 = 1+1$

$= \boxed{2}$

1×4

Declaration & Initialization

int a[ ]; Invalid

int a[ ] = {10, 20, 30}; ✓

int a[10];

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

① If we are declaring an array without initialization, then it is mandatory (compulsory) to provide the size of each dimension, otherwise error would be there.

1) int a[ ] ; Invalid

2) int a[2] ; ✓

3) int a[ ][ ] ; ✗

4) int a[ ][3] ; ✗

5) int a[2][3] ; ✓

6) int a[ ][ ][ ] ; ✗

7) int a[10][ ][ ] ; ✗

8) int a[2][3][ ] ; ✗

9) int a[2][3][2] ; ✓

② In case we are initialiazing an array, then we have the flexibility that we can omit 1st dimension size but no other dimension is having such flexibility.

(i) `int a[] = {10,20};` ✓

(ii) `int a[ ][3] = {1,2,3,4};` ✓

(iii) `int a[2][ ] = {1,2,3,4};` ✗

(iv) `int a[ ][ ] = {1,2,3,4};` ✗

→ flexibility सिर्फ 1st dim. के लिए है।

② In case we are initialiozing an array, then we have the flexibility that we can omit 1st dimension size

but no other dimension is having such flexibility.

अगर हम array का initialization कर रहे हैं तो 1st dim का size नहीं भी देंगे तो Compiler हमें Tiger भईया की तरह ud kee laat नहीं मारेगा।

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

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

$$x \times 3 = 4$$

$$x = \frac{4}{3} = \lceil 1.33 \rceil = 2$$

for compiler

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

1	2	3	4	0	0

Pointers

A pointer variable is a special variable that is used to hold the address of other variable.

int a ;   simple variable

special variable

int *p ;

int x;

float y

int *p;

Pointer to

int * p ;

is a

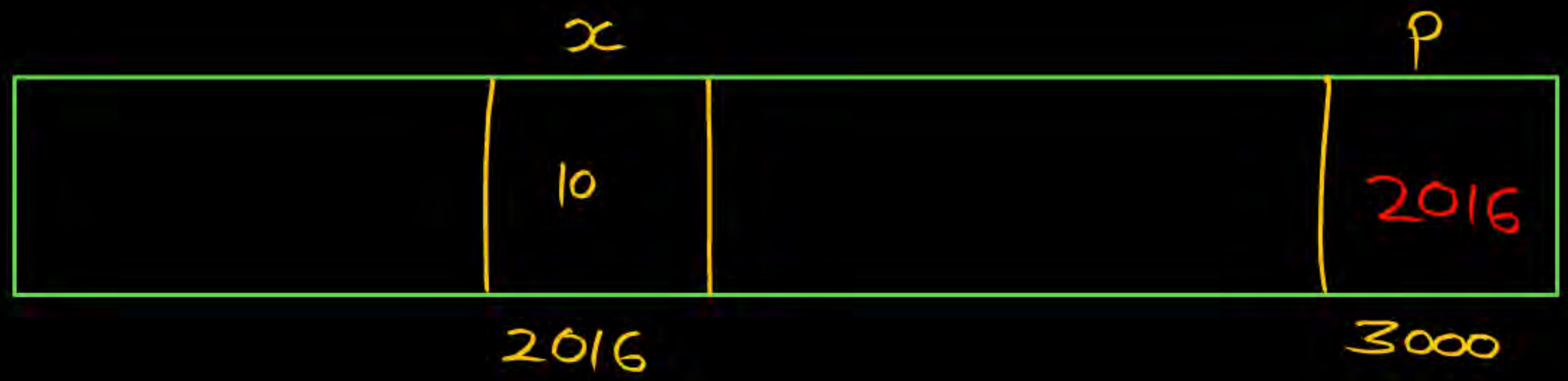
P is a pointer to integer.

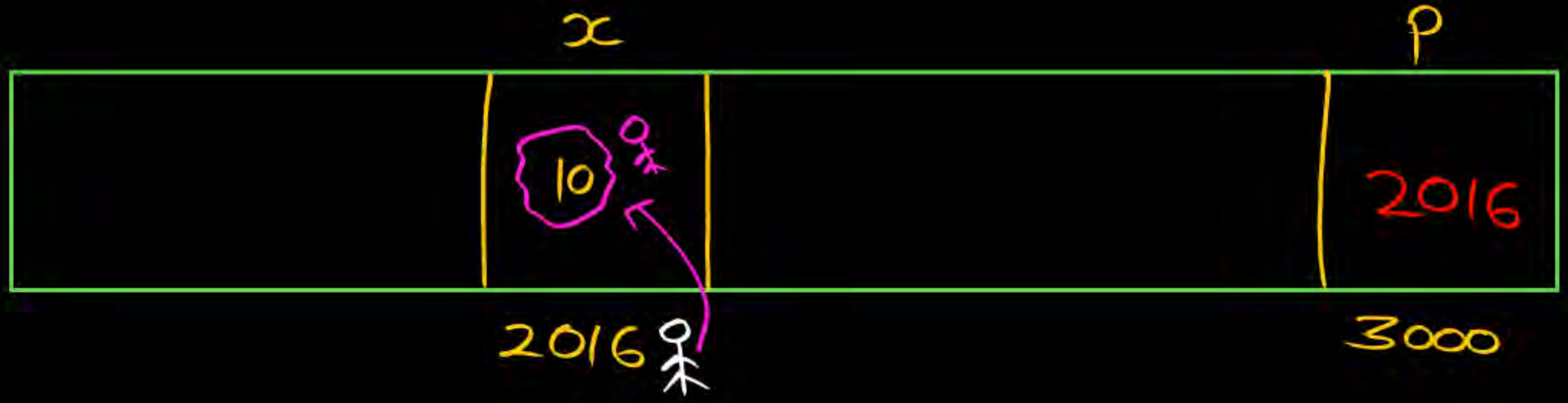
P can store address of some integer variable.

```
int x = 10;
int *p;   p के अन्दर
```

⇒ address
of
any integer
variable

```
p = &x ;
```





x ⇒ 10

p ⇒ (Memory location 2016)

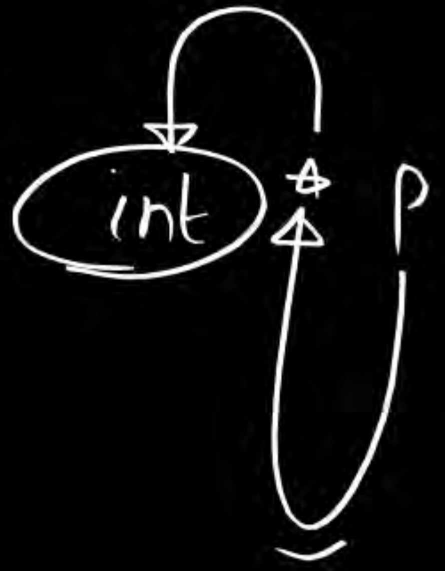
*p ⇒ value at (Memory location 2016) ⇒ 10

```
printf("%d", &x);
printf("%d", p);    ] 2016
printf("%d", *p);     10
```

int x = 10;

int *p; → int.

p = &x

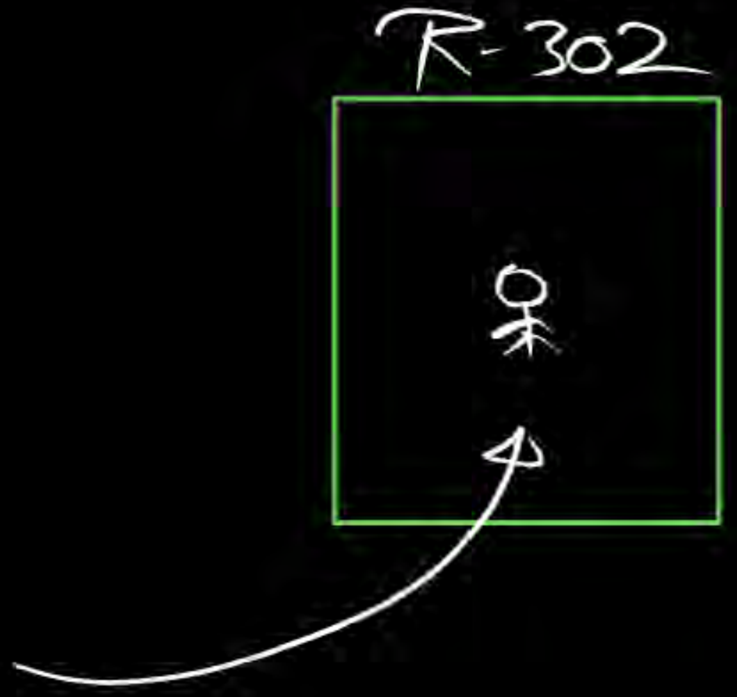


```
int  x ;

int *p ;

int * * q
```

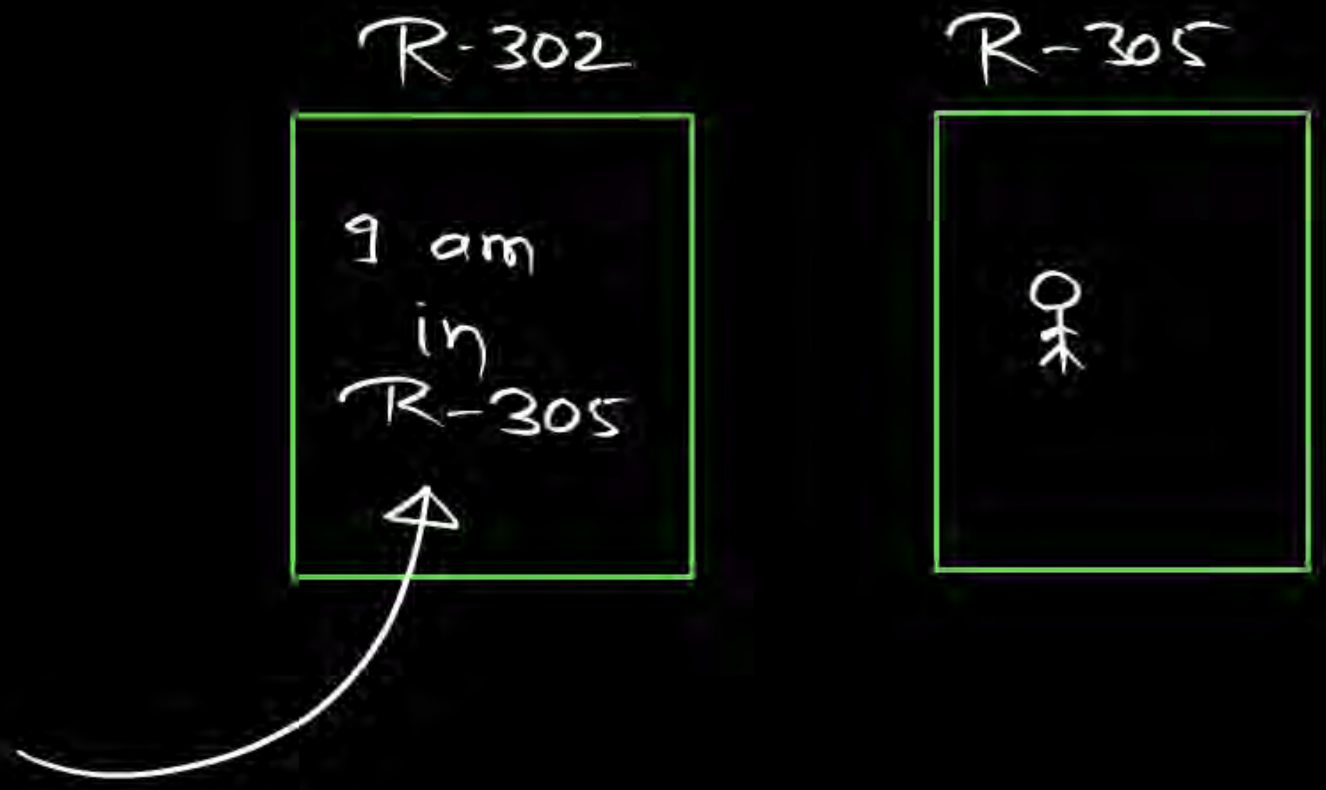
Pointer to int का

int x;

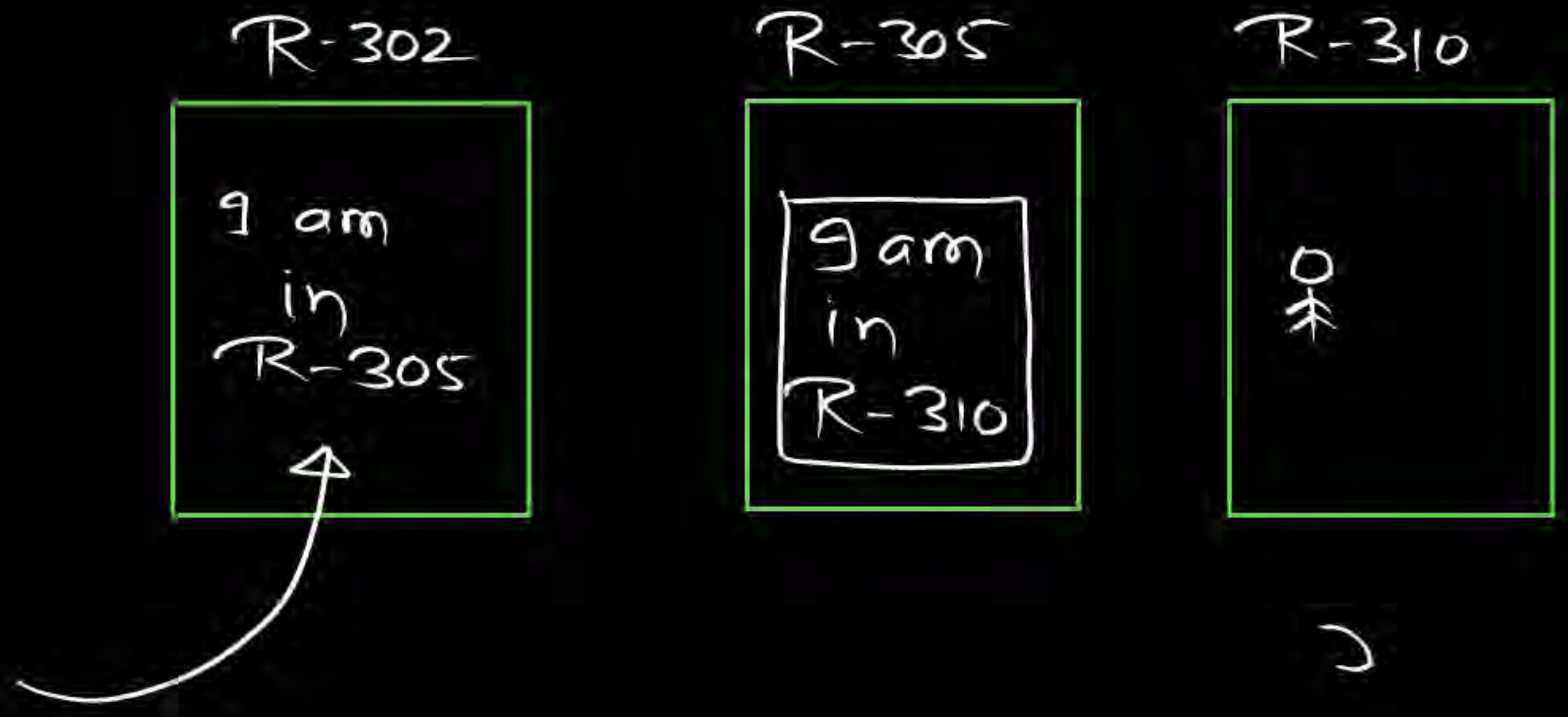
offline



```
int x;

int *p;
```



```
int x;

int *p;

int **q;
```



```
int x ;
int *p ;
int **q ;
```

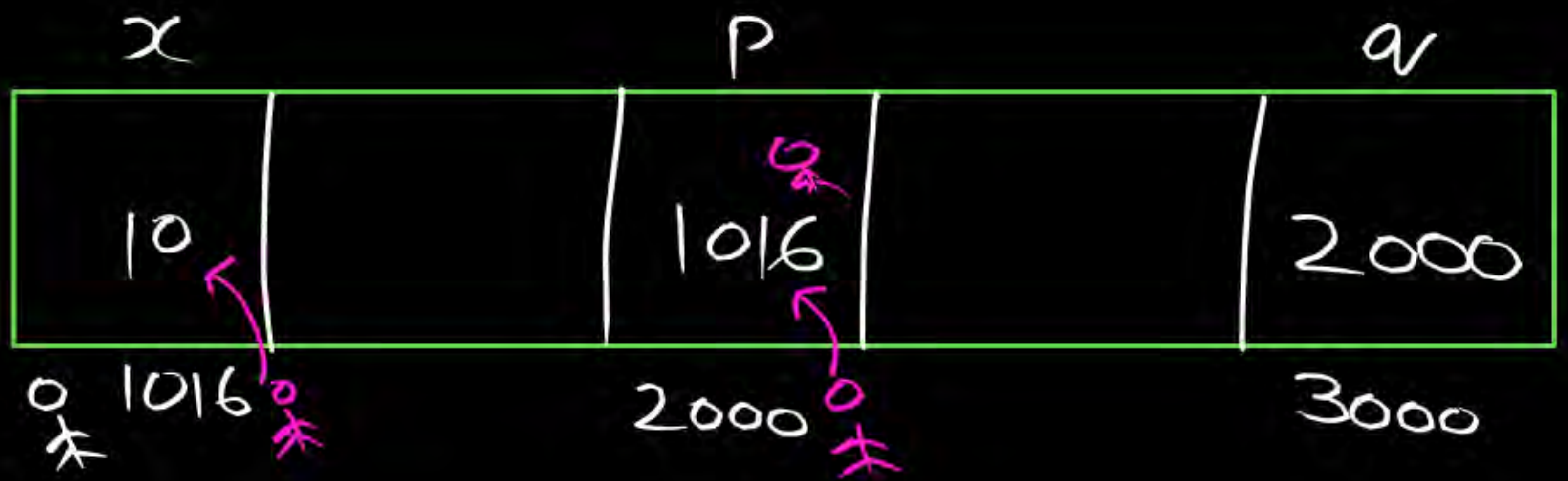
```
q = &x ;
```

logically incorrect

```
q = &p ;
```

✓

```
int x = 10;
int *p;
int **q;
p = &x;  ✓
q = &p;  ✓
pf("%d", x);      10
pf("%d", p);   ⇒ 1016
pf("%d", *p);  ⇒ 10
pf("%d", q);   ⇒ 2000
pf("%d", *q);  ⇒ 1016
pf("%d", **q); → 10
```



$p = \overset{\text{Mem. loc.}}{1016}$

$*p = \text{value at}\left(\text{Mem. loc. } 1016\right)$

$= 10$

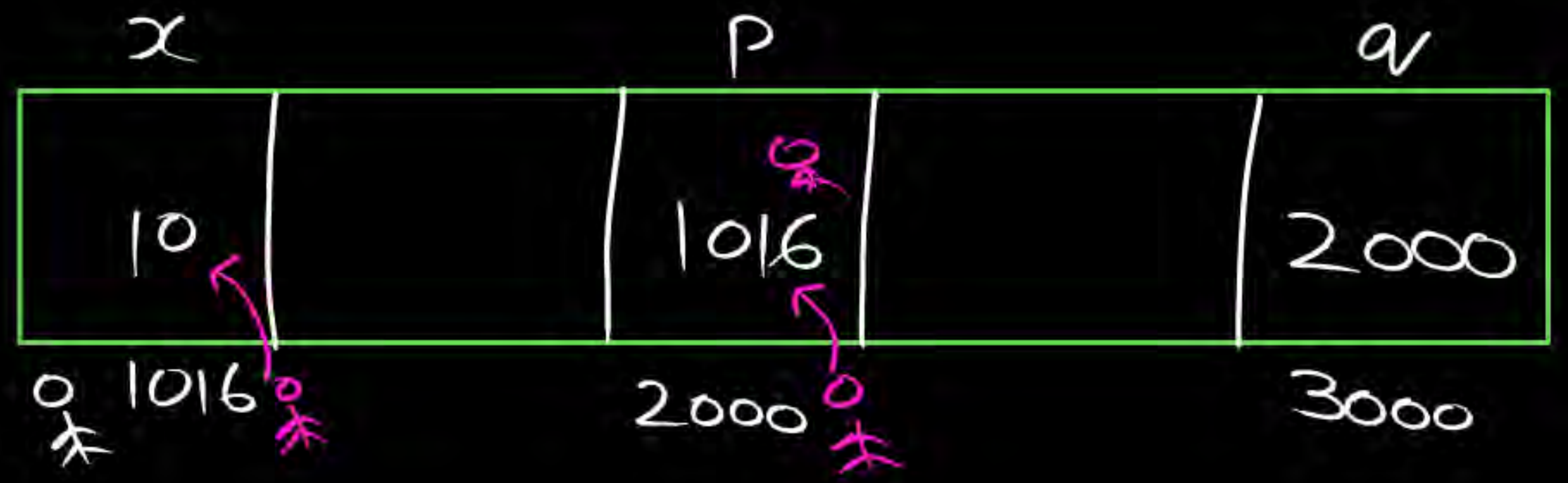
$q \Rightarrow \text{Mem. loc. } 2000$

$*q \Rightarrow \text{value at}\left(\text{Mem. loc. } 2000\right)$

$*q = \text{Mem. loc. } 1016$

$**q = \text{value at}\left(\text{Mem. loc. } 1016\right) = 10$

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



$p = \&x$

$*p = \cancel{*}\cancel{\&}x \; ; \Rightarrow *p = x$

$q = \&p$

$*q = \cancel{*}\cancel{\&}p = p$

$*q = p = \&x$

$**q = *p = \cancel{*}\cancel{\&}x = x = 10$

THANK YOU GW SOLDIERS !