unacademy

Arrays & Pointers Part - III

Comprehensive Course on C- Programming

Pankaj Sharma • Lesson 21 • May 28, 2024

unacademy

CS & IT Engineering

C Programming

Arrays & Pointers-III

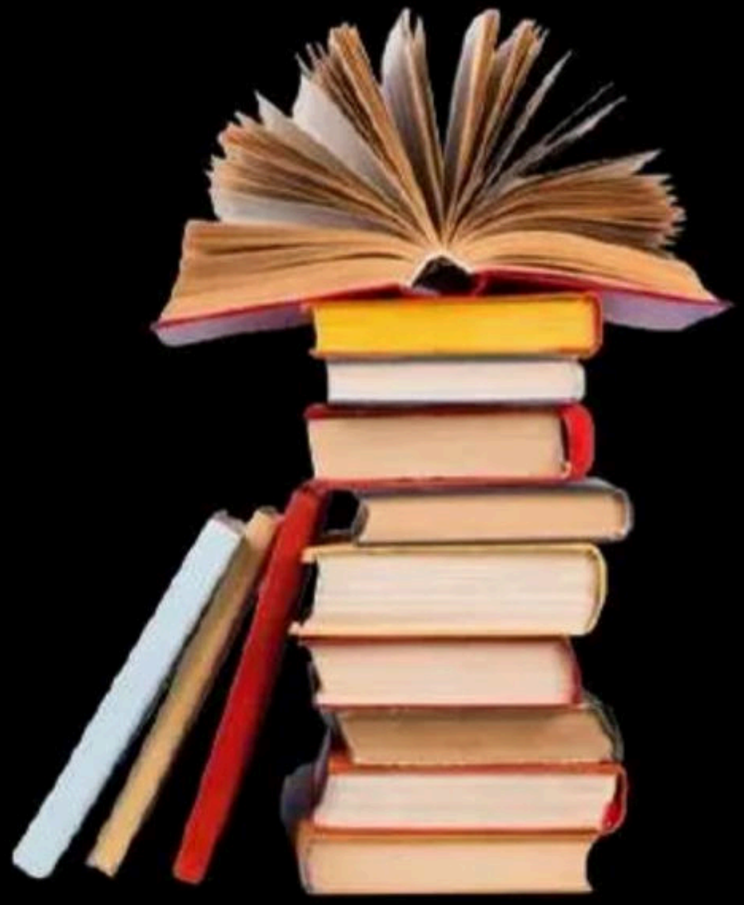
Lecture Number- 21

By- Pankaj Sir

unacademy

Topics

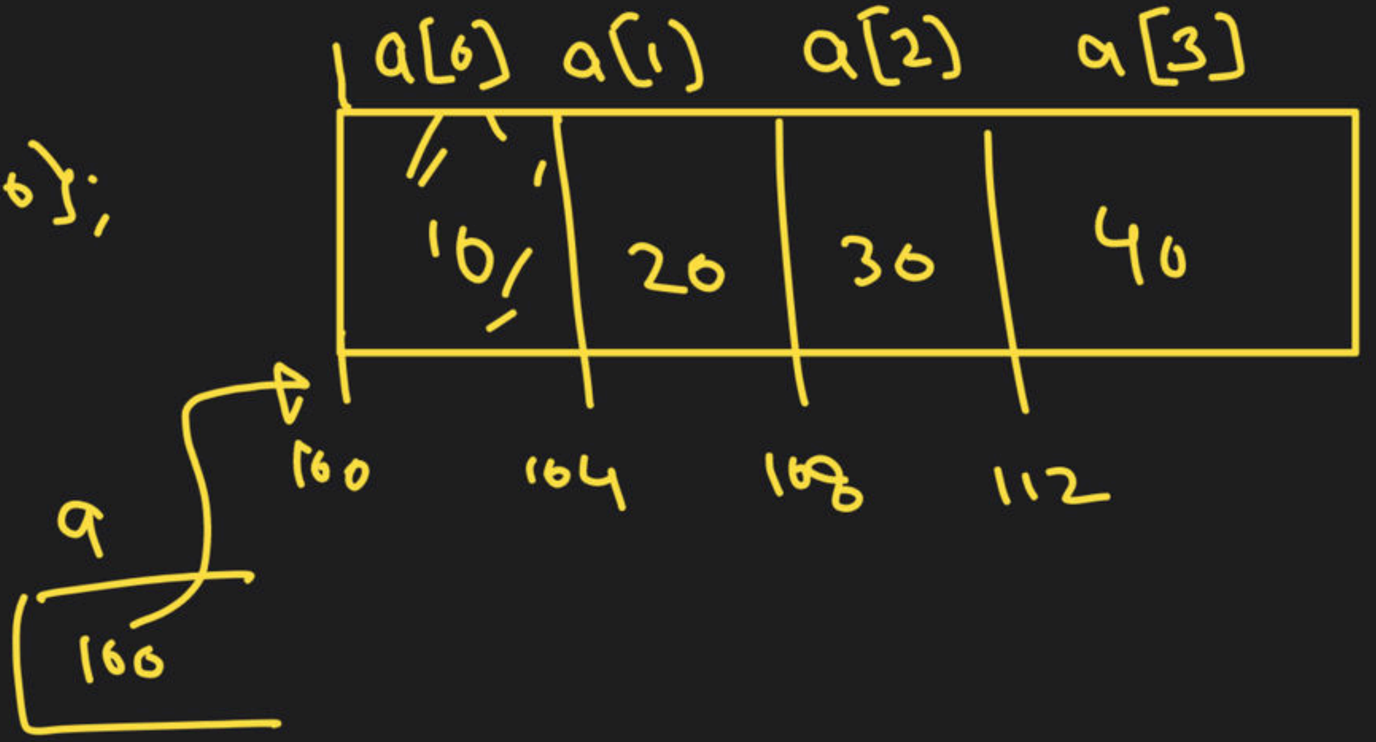
to be covered



1. Arrays & Pointers Part-III

```
void main() {
    int a[4] = {10,20,30,40};
    printf("%u", a);
    printf("%u", &a[0]);
}
```

a ≡ &a[0]



→ Name of Array does not represent an address with 2 operators

a) address of operator (&)

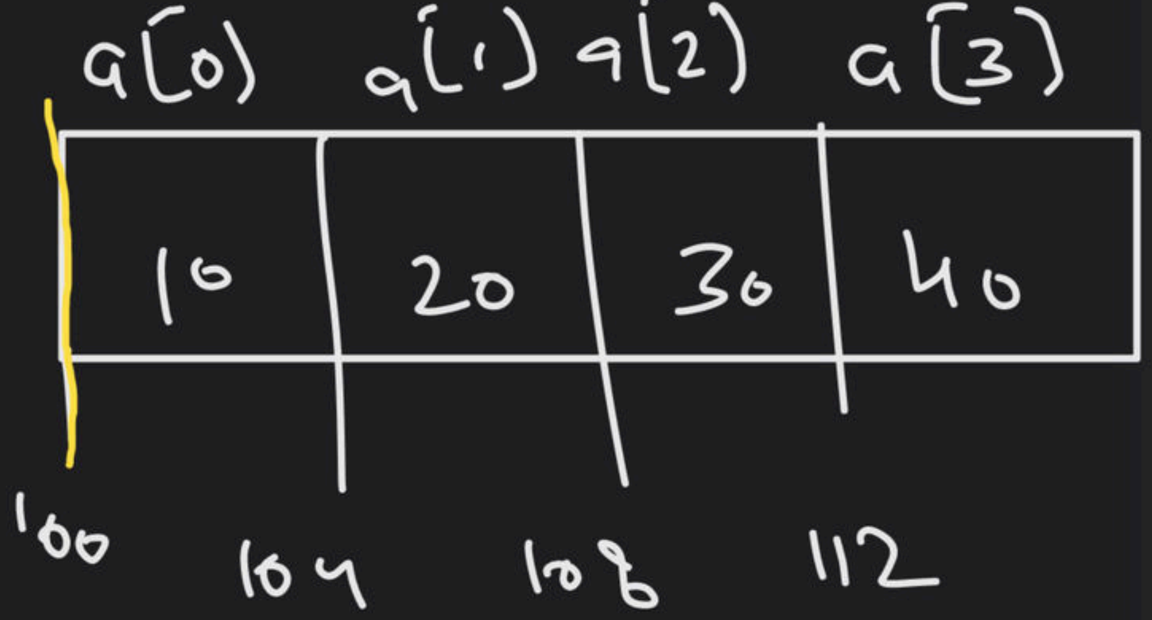
b) sizeof operator

unacademy

```
int a[4] = {10,20,30,40};
```

$a \equiv \&a[0]$

$\&a \rightarrow$ address of whole array



unacademy

```
int a[4] = {10,20,30,40};
printf("%u", a);
```

→ addr. of first element

1000

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

→ 1000

→ 1000

a[0]	a[1]	a[2]	a[3]
10	20	30	40
1000	1004	1008	1012

Numerical value is same

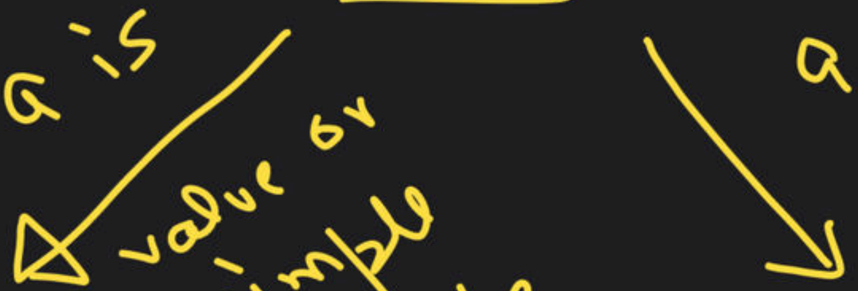
16 byte

4 byt

unacademy

Numerical value of a is 100

a+1 ?




```
int a = 100;
pf("%d", a+1);
```

101

value + val = val (arith.)

what is a

address arithmetic.

add + val ⇒ add

val + add ⇒ add

add + add ⇒ Invalid

* If declaration of array is having n-dimensions And

a) Anywhere in program, u provide exactly n-dimension then it is an element.

b) Anywhere in program, u provide less than n-dimension then it is ⇒ address.

```
int a[4]={10,20,30,40};
```

=

a → 0-dim

a → addr ✓
a → elem ✗

a[0] → elem ✓
a[0] → addr ✗

2-dim

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

a → addr ✓ (0-dim)
a → elem ✗

a[0] → addr ✓ (1-dim)
a[0] → elem ✗

a[1] → addr ✓
a[1] → elem ✗

a[1][1] → addr ✗

```
int a[3][3][3];
```

a →
a[0]
a[0][1]
} address

a[0][0][0] → elem

a + 1

1) What is a? → value → dance
→ address

2) Kiska address hai (whose address we are talking about)

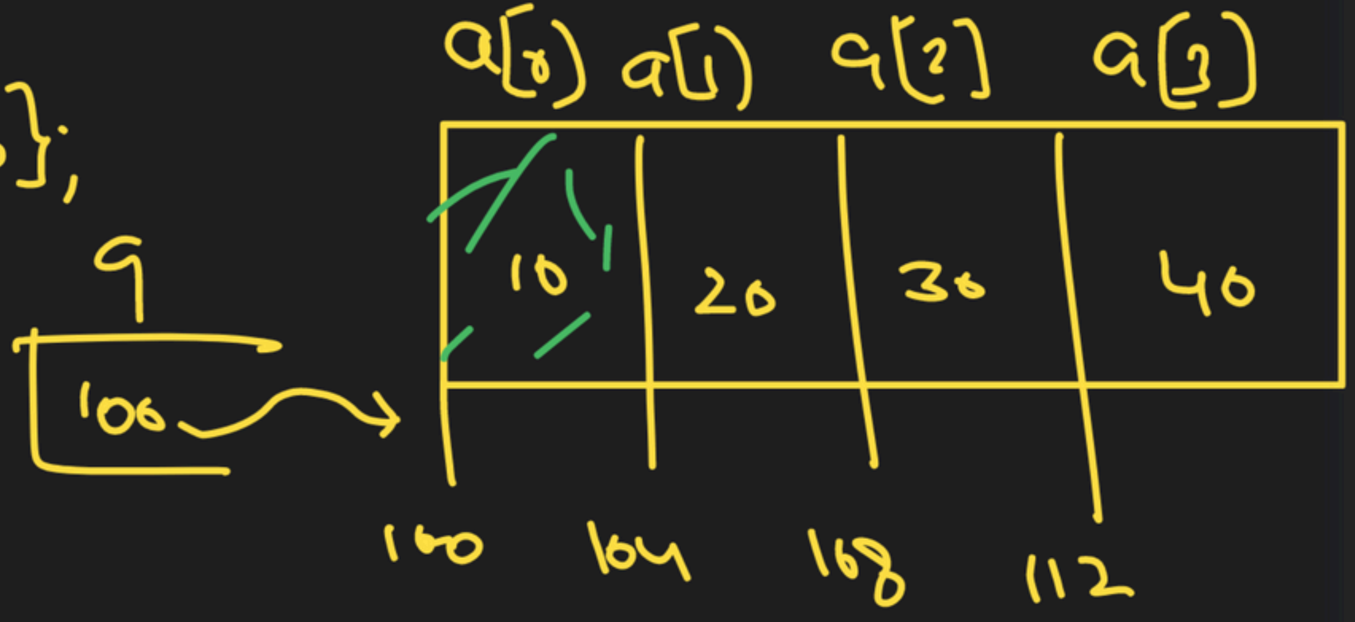
3) Uska size kya hai (Find the size of that Object)

unacademy

```
int a[4] = {10,20,30,40};
```

a+1

(i) a → addr
a → ele x



(ii) Kisska? whose add.

a → array ka naam ⇒ Add. of its first element
→ address of a[0]

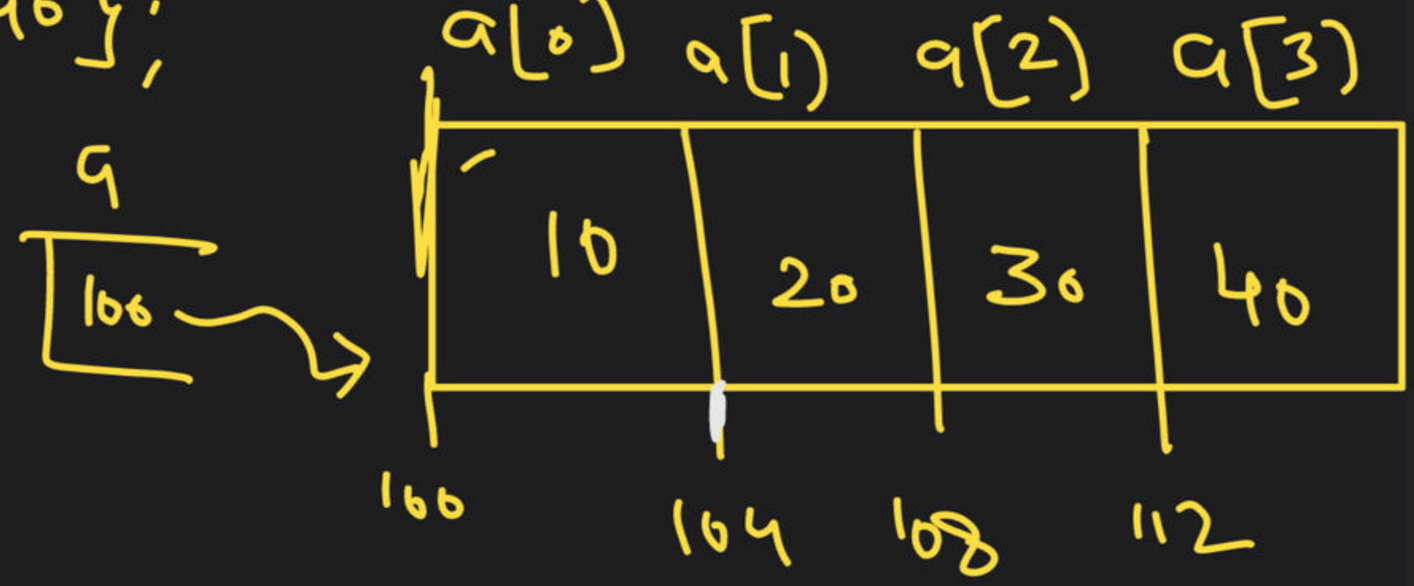
iii) size of a[0] ⇒ 4 bytes

unacademy

```
int a[4] = {10,20,30,40};
```

a+1

&a[0]+1x4



⇒ 100+4 = 104

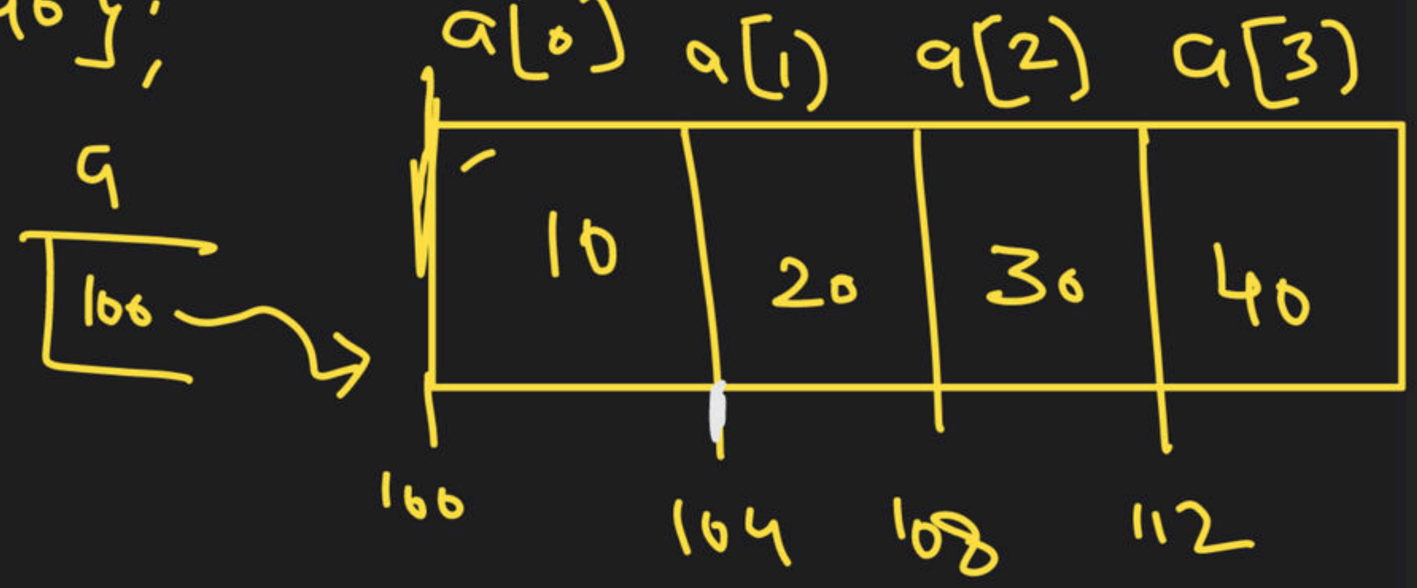
a+1 = &a[0]+1x4 = 100+4 = &a[1]

(a+1) = (Memory loc. 104) = &a[1]

unacademy

```
int a[4] = {10,20,30,40};
```

a+2



(i) a → addr. ✓ / else ✗

(ii) a ≡ &a[0]

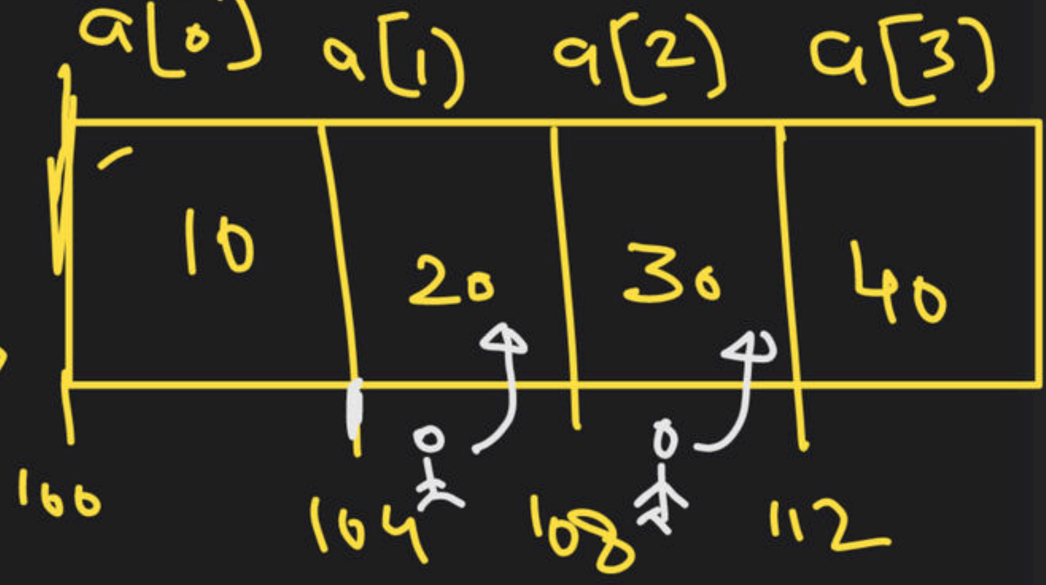
(iii) sizeof a[0] ⇒ 4 bytes

a+2 ⇒ &a[0] + 2×4

⇒ 100 + 8 = 108

a+2 = (Memory loc. 108) ≡ &a[2]

```
int a[4] = {10,20,30,40};
```



$a+1 \equiv (\text{Mem. location } 104) = \&a[1]$ —(1)

$a+2 \equiv \text{Mem. loc. } 108 = \&a[2]$ —(2)

$*(a+1) = \text{value at}(\text{Mem. loc. } 104) = *\&a[1]$

$*(a+2) = \text{value at}(\text{Mem. loc. } 108) = *\&a[2]$

$*(a+1) = 20 = a[1]$

$*(a+2) = 30 = a[2]$

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

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

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

```
void main(){
    int a[4] = {10,20,30,40};
    printf("%d", a[1]);
    printf("%d", *(a+1));
}
```

20

Addition commutative

$2 + 3 \Leftrightarrow 3 + 2$

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

$= *(i+a)$

$= i[a]$

#

```
void main(){
    int a[4] = {10,20,30,40};
    printf("%d", a[2]);
    printf("%d", *(a+2));
    printf("%d", *(2+a));
    printf("%d", 2[a]);
}
```

unacademy

declaration

int a[a] = {10,20,30,40};

Compiler info. Save

```
void main(){
int a[5] = {10,20,30,40,50};
pf("%u", a);
pf("%u", &a);
pf("%u", a+1);
pf("%u", &a+1);
pf("%u", *(a+1));
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
10	20	30	40	50
1000	1004	1008	1012	1016

→ add/ele → add. of first ele ⇒ &a[0] → 1000

→ add. of whole array → 1000

→ a → add → &a[0] → 4 bytes

a+1 = &a[0] + 1×4 ⇒ 1000 + 4 ⇒ 1004

(i) &a → add. of whole array (26 byte)

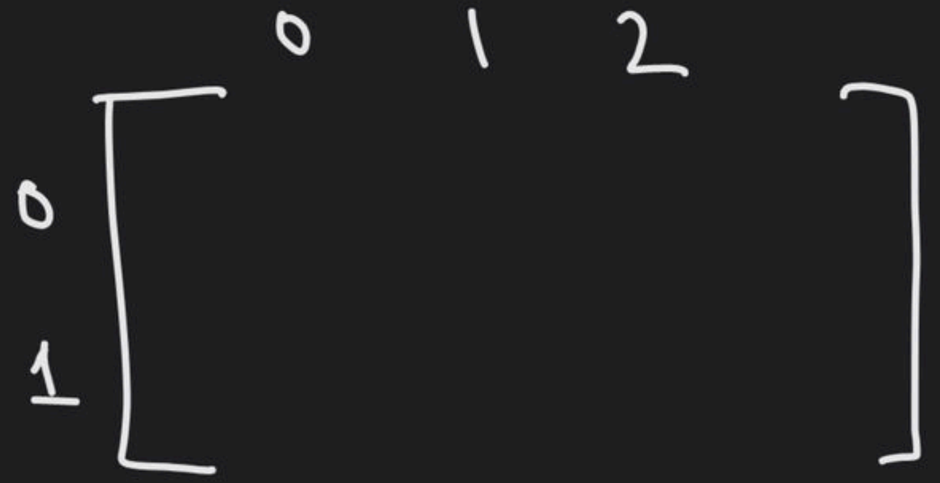
&a+1 ⇒ &a + 1×20 ⇒ 1000 + 20 = 1020

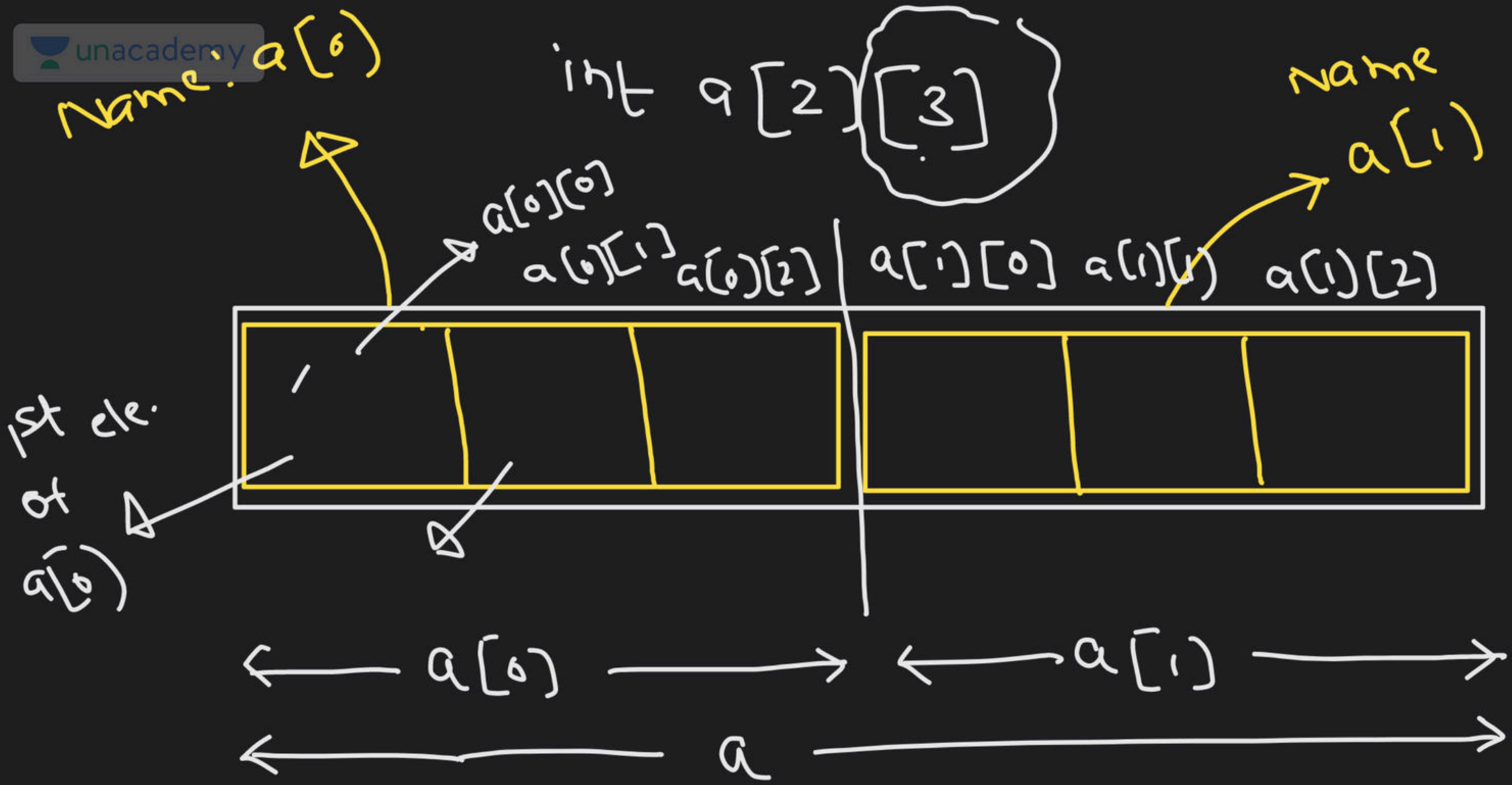
→ a[1] ⇒ 20

unacademy

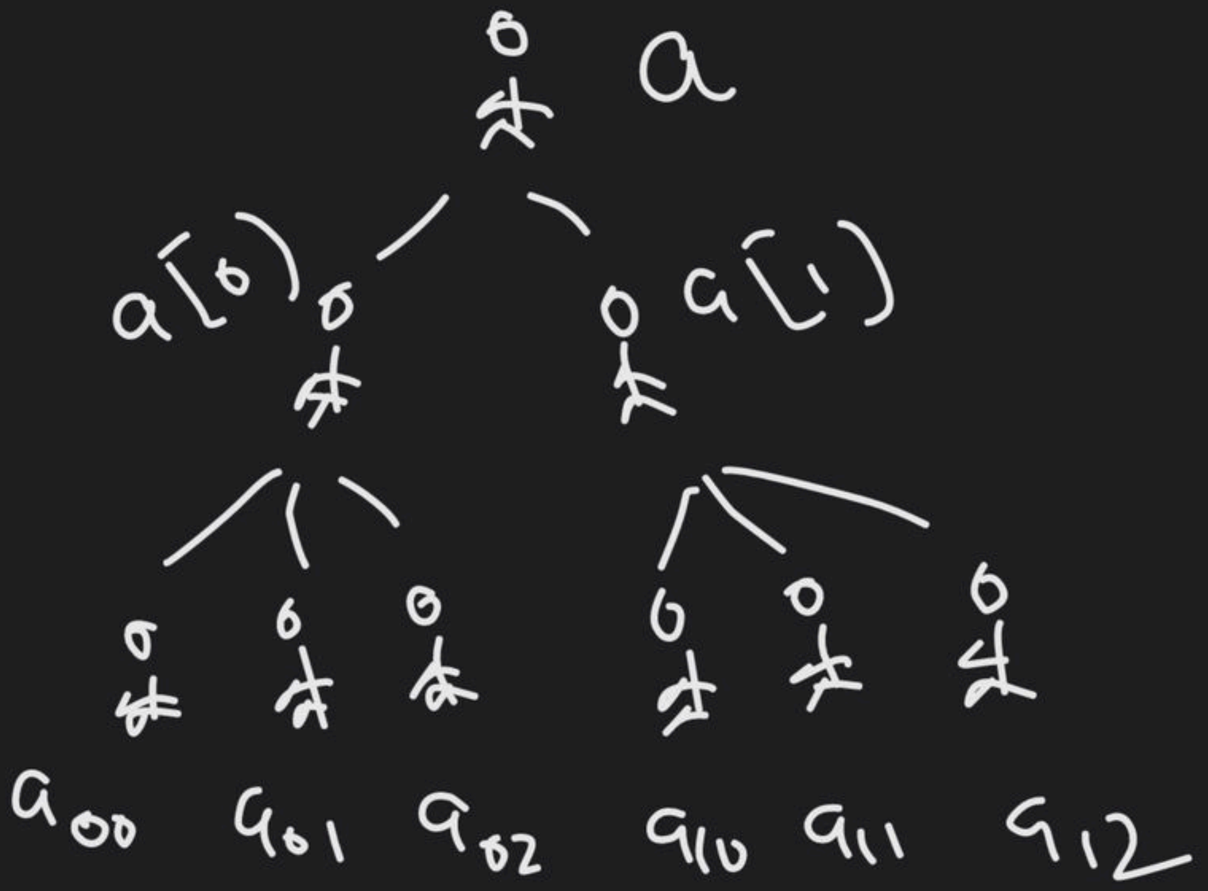
2-D array

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




Name: a[0]
int a[2][3]
Name a[1]
a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]
1st ele. of a[0]
a[0]
a[1]
a

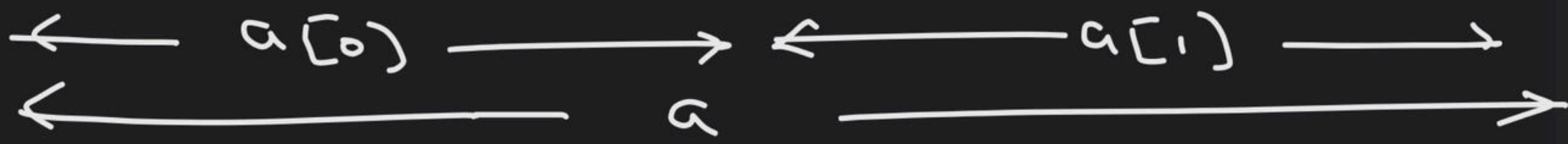
a[2][3]



int a[2][3] = {10,20,30,40,50,60};

(1) a ⇒ address of 1st element ⇒ Kiska

(2) a[0] ⇒ address of 1st element



(1) a ⇒ &a[0] (12 byte)

(2) a[0] → &a[0][0] (4 byte)

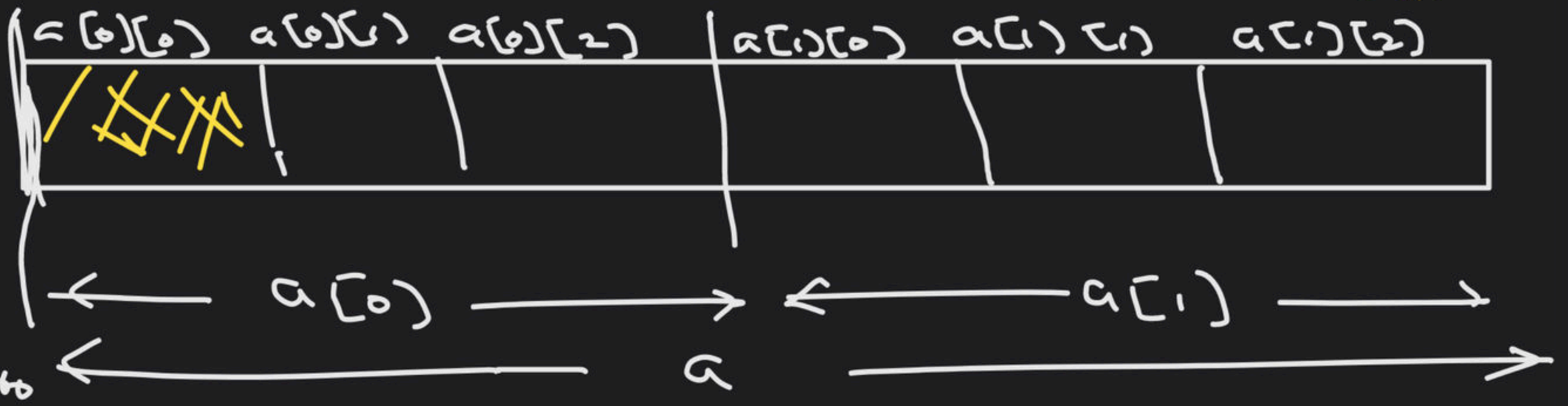
(3) &a → Address of whole array (24 byte)

a[0] ⇒ array of 3 elem → 100, 101, 102	a → collection (array) of 2 elem. ⇒ i.e. a[0], a[1]

int a[2][3] = {10,20,30,40,50,60};

① a → address of ele ⇒ kiska

② a[0] → address of ele



pf("%u", a);

100

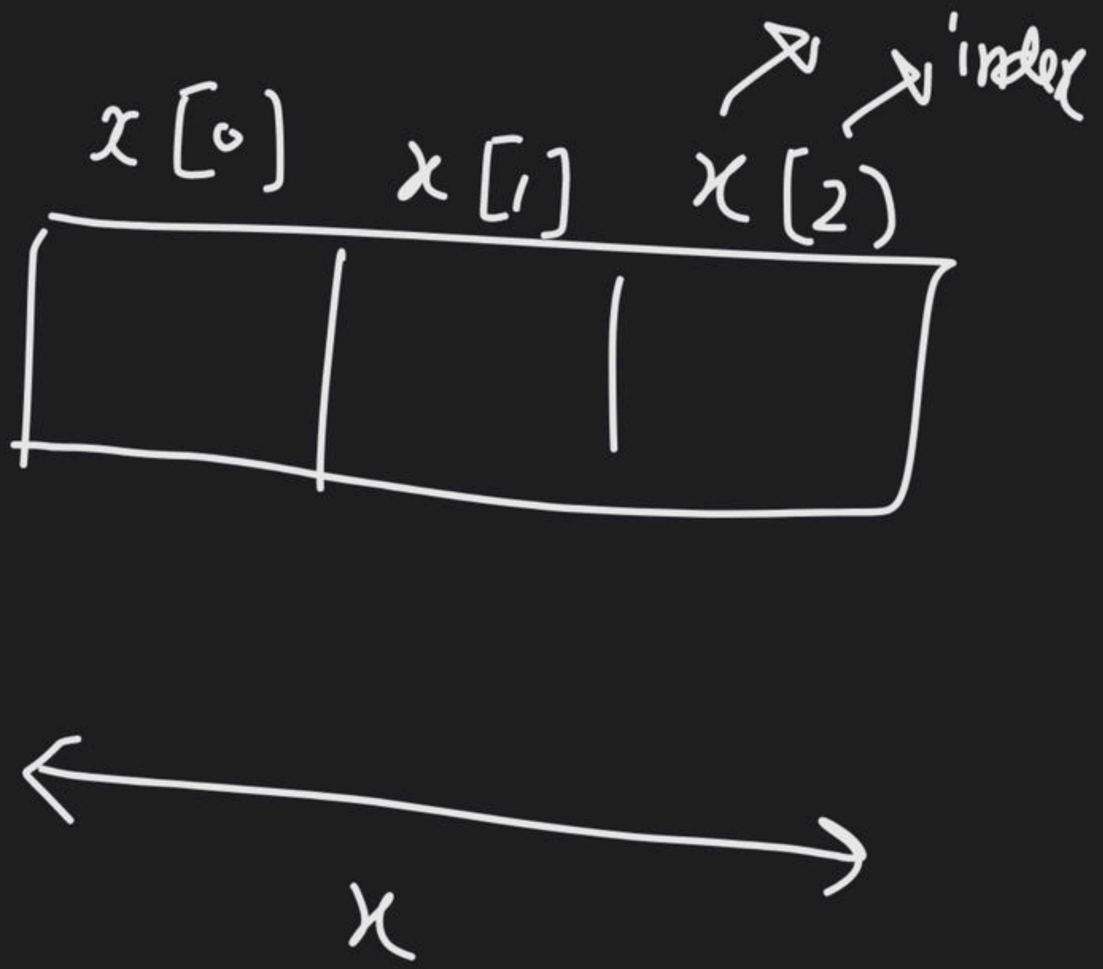
② pf("%u", a[0]) → address of ele x ⇒ &a[0][0]

a[0] ⇒ array of 3 elem

a_{00}, a_{01}, a_{02}

③ pf("%u", &a); → 100 → address of whole array

100



unacademy

```
int a[4];      → 1-D
int a[3][4];   → 2-D
int a[2][2][3];
```

→ 3-D

unacademy

THANK YOU!

Here's to a cracking journey ahead!