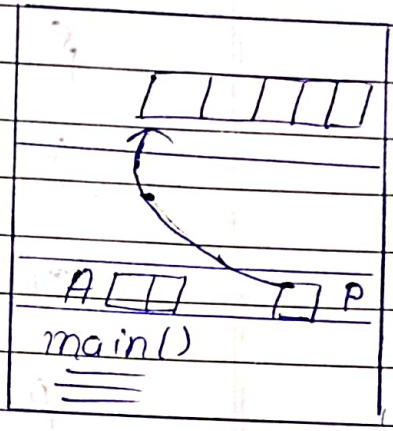
Date / /
Page No.

Array

Static V/s Dynamic Array

```
Void main ()
{
    int A[5];
    // Can use VLA
    int *p;
    p = new int[5];          // In C++
    p = (int*) malloc (5 * sizeof (int))
    . . .                    // In C
    . . . . . .
    delete [] p;             // In C++
    free (p);                // In C
}
```



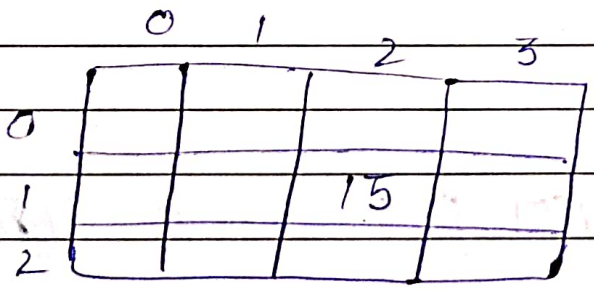
Date / /
Page No.

* Increasing Array size

```
int *p = new int[5];
int *q = new int[10];
for(int i=0 ; i<5; i++)
{
    (q[i]) = (p[i]);
}
delete []p;
p = q;
q = NULL;
```

* 2-D Array

M-1 : `int A[3][4] = {{1,2,3,4},{2,4,6,8},{3,5,7,9}}`

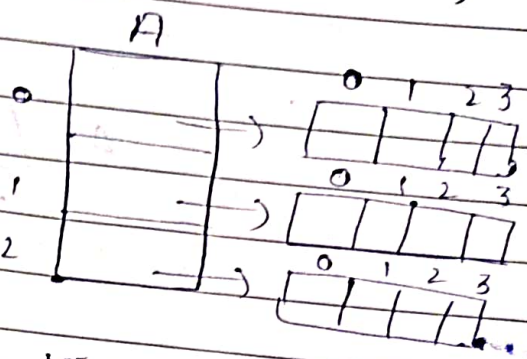


A[1][2] = 15

Date / /
Page No.

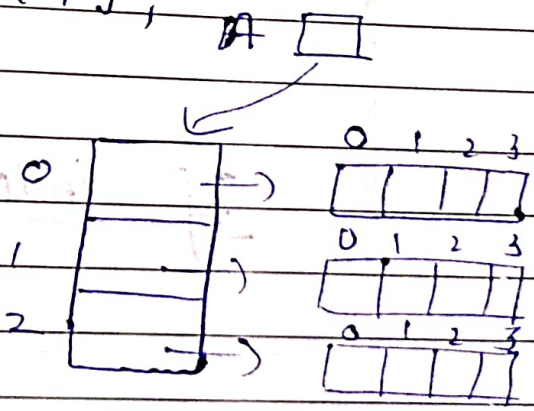
M-2 `int * A[3]` { Array of Integer pointers }

```
A[0] = new int [4];
A[1] = new int [4];
A[2] = new int [4];
A[1][2] = 15;
```



M-3 { Everything will be in heap }

```
int ** A;
A = new int * [3];
A [0] = new int [4];
A [i] = new int [4];
A [2] = new int [4];
```

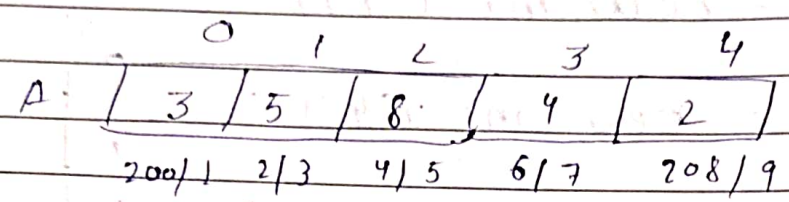


{ for accessing, we have to use two for loops }

Date / /
Page No.

* Array in Compilers

int A[5] = {3, 5, 8, 4, 2}



A[3] = 10;

Addr(A[3]) = 200 + 3 * 2 = 206

Addr(A[3]) = Lo + 3 * 2 = 206
↳ if Lo = 200

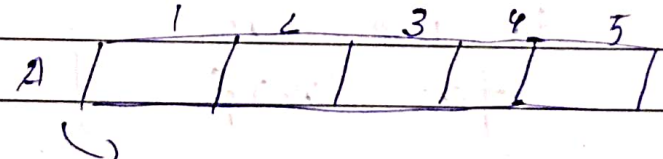
Addr(A[i]) = Lo + i * sizeof(data-type)

Addr(A[i]) = Lo + i * w

Lo → Base Addr
i → index
w → Size of data type

If index starts from 1 :-

int A[1...5]



A[3] = 10;

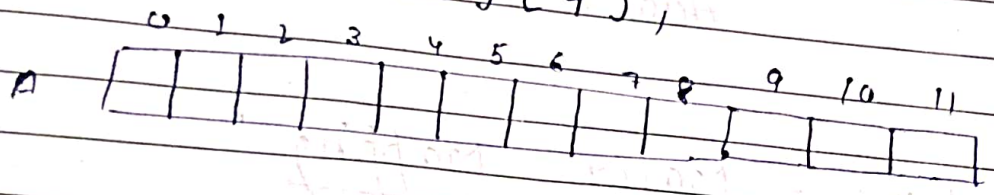
Date / /
Page No.

Addr (A[3]) = 200 + (3-1) * 2 = 204

Addr (A[i]) = 200 + (i-1) * w

* 2D Array in Compiler

int A[3][4];



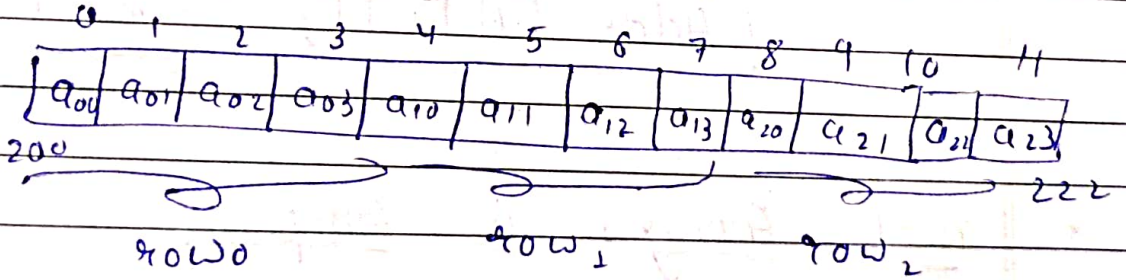
Representation :-

A	0	1	2	3
0	a_{00}	a_{01}	a_{02}	a_{03}
1	a_{10}	a_{11}	a_{12}	a_{13}
2	a_{20}	a_{21}	a_{22}	a_{23}

1. Row - major Mapping
2. Column - major Mapping.

(1) Row - Major Mapping :-

int A[3][4];
m n



Add (A[1][2]) = 200 + [4 + 2] * 2

= 200 + 6 * 2 => 212

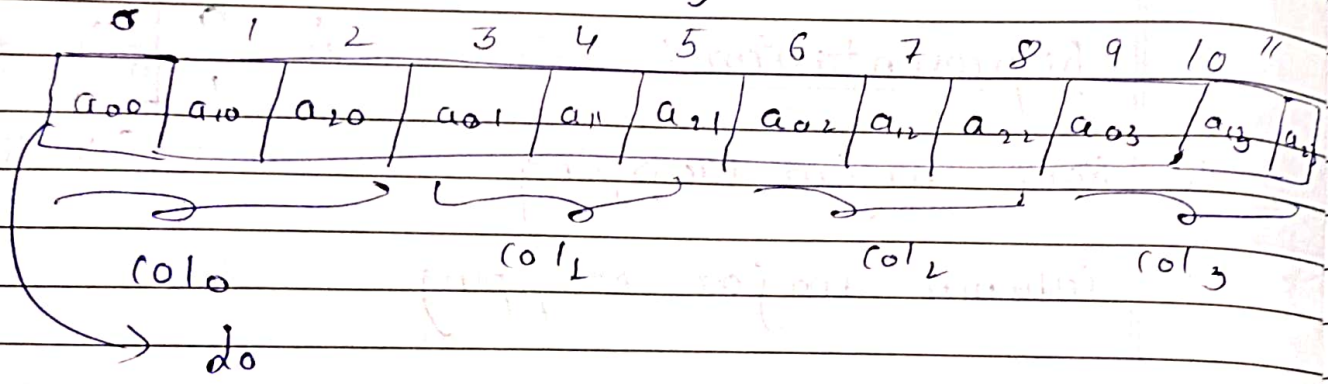
Add of (A[2][3]) = 200 + [2 * 4 + 3] * 2
⇒ 222

∴ Addr [A[i][j]] = α_0 + [i * n + j] * w

If array starts from 1 onwards :-

Addr [A[i][j] = α_0 + [(i-1) * n + (j-1)] * w

② Column - Major Mapping :-



Addr (A[1][2]) = 200 + (2 * 3 + 1) * 2

Addr [A[i][j]] = α_0 + (j * m + i) * w

* n-D Arrays in compiler

Type A[d_1][d_2][d_3][d_4]

Date / /
Page No.

Row - Major :-

$$Addr(A[i_1][i_2][i_3][i_4]) = d_0 + [i_1 * d_2 * d_3 * d_4 + i_2 * d_3 * d_4 + i_3 * d_4 + i_4] * w$$

Column - major :-

$$Addr(A[i_1][i_2][i_3][i_4]) = d_0 + [i_4 * d_1 * d_2 * d_3 + i_3 * d_1 * d_2 + i_2 * d_1 + i_1] * w$$

Generalising :-

$$\text{Row - Major} = L_0 + \sum_{p=1}^{n} \left(i_p * \prod_{q=p+1}^{n} d_q \right) * w$$

$$\text{Column - Major} = L_0 + \sum_{p=n}^{1} \left[i_p * \prod_{q=p-1}^{1} d_q \right] * w$$

Multiplications :

4D → 3 + 2 + 1

5D → 4 + 3 + 2 + 1

nD → (n-1) + (n-2) + ... 1

$$\Rightarrow \frac{n(n-1)}{2} = O(n^2)$$

Date / /
Page No.

Reducing the no of multiplications :-

HORNER'S RULE

$\Rightarrow i_4 + i_3 * d_4 + i_2 * d_3 * d_4 + i_1 * d_2 * d_3 * d_4$

$\Rightarrow i_4 + d_4 * [i_3 + i_2 * d_3 + i_1 * d_2 * d_3]$

$\Rightarrow i_4 + d_4 [i_3 + d_3 [i_2 + i_1 * d_2]]$

① ② ③

No of Multiplications = 3

$O(n)$

4D → 3
5D → 4
nD → n-1