MULTI DIMENSIONAL ARRAY

* Multi-dimensional array does not orignally exist in memory. It is just a perception for the easiment of user.

* We map multi-dimensional array with the memory of 1-D array.

* A multi-dimensional array is an array having more than one dimensions.

* To refer elements in such arrays more, then one subscripts / indices are needed

* The subscripts / indices are based on number of dimensions;
e.g. to refer **2D** array, two indices are required. Similarly, for **3D** array, three indices are required and so on.

2-D ARRAY

* Two dimensional arrays are also termed as **Matrices** in which elements are ordered in number of **rows** and **columns.**

* An example of **m x n** matrix (2D array) is

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{matrix}$$

* `int A[3][5];`
  A **2-D** Array of **3** rows and **5** columns

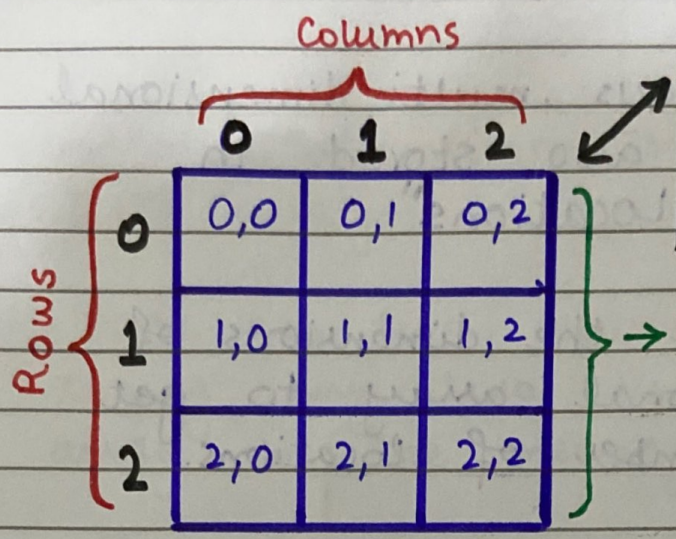
* A **2-D** Array is an array of **1-D** arrays.

* In the above example, each row (0, 1, 2) has **1-D** array of **5** elements.

* To refer an element of **2D** array, we have to specify the **row number** (first dimension) and **column number** (second dimension).

EXAMPLE OF 2-D ARRAY:

```
int A[3][3];
```



It is for the easyment of user. In actual, 2D array does not exist. It only maps with 1-D array.

→ It will first find dimensions.

D_1 (Row) = 3

D_2 (Column) = 3

★ Bcz 2D Array orignally does not exist, that's why we find dimensions.

then

Size = Row X Column

= 3 x 3

Size = 9

Multi Dimensional Array Memory Representation:

* Like **1D** arrays, multi dimensional arrays are also stored in "Contagious Locations".

* Multiply all the dimensions of multi dimensional array to get required number of locations.

* In array **int A[3][5]**, the total number of locations required are **3 x 5 = 15**

* Similarly in a **3D** array **int[3][5][4]** the required number of location will be **3 x 5 x 4 = 60.**

2-D Array Memory Representation:

★ To map Multi-dimensional array in 1-D array:

→ There are two conventions of storing any two dimensional array in the memory.

(1) Row-Major Order:

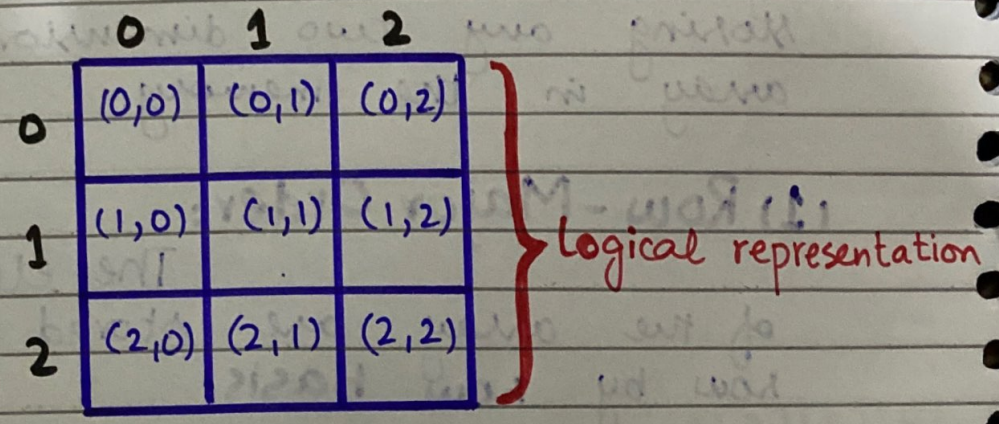
The elements of the array are stored on a row by **row basis.**

(2) Column-Major Order:

The elements of the array are stored on a column by **column basis.**

Row Major Order

* The elements of the array are stored on a row by row basis, i.e. all the elements of first row, then in the second row, and so on.



It will first find Table dimensions.

D_1 (Row) = 3

D_2 (column) = 3

then,

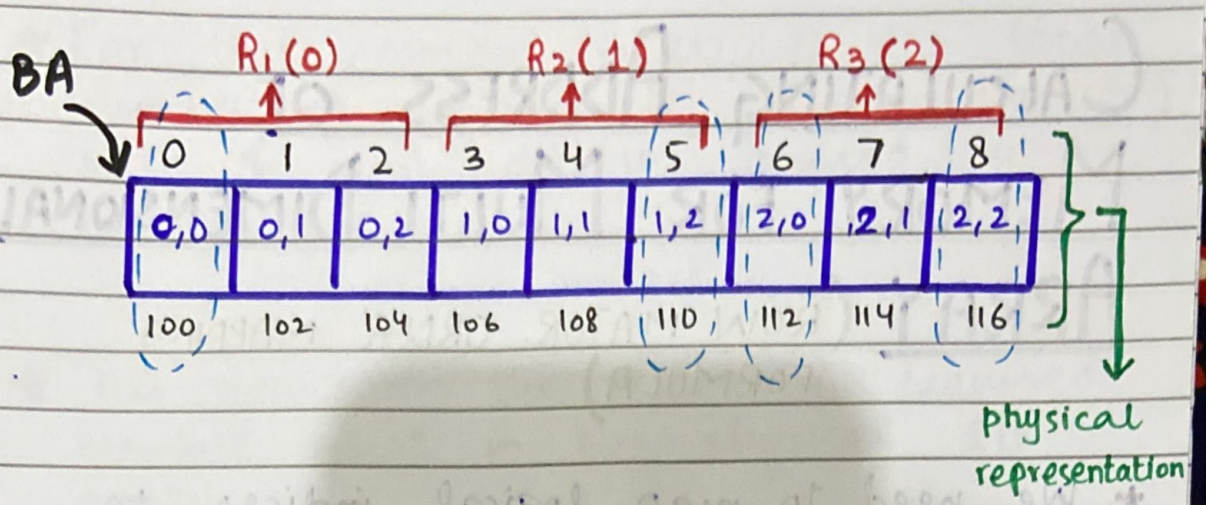
Size = Row × Columns

= 3 × 3 = 9

So,

Upper Bound = 8

Lower Bound = 0



NOTE :

★ In row major order, it will place data in physical memory 'row by row'.

So,

MD → 1D → Array

Calculating Address of Memory for Multi Dimensional Array (Row Major Order Mapping Formula)

* We need to map logical indices to physical indices to access a particular element of the array.

* Suppose a **2D** array has u_1 **rows** and u_2 **columns**, an element in the i^{th} **row** and j^{th} **column** can be obtained by the following formula:

General Formula

$$D_1 = u_1$$
$$D_2 = u_2$$

* $A[u_1][u_2]$

$$Address(i,j) = BA + (i u_2 + j) \times ES$$

($i u_2$: $A[u_1][u_2]$ no. of col. to skip; $(i u_2 + j)$: index of 1D)

* To go to a specific row (i^{th}) we have to skip that specific number of columns (u_2).

* For this, simply multiply the total number of columns u_2 (second dimension) with **i** and add the desired column number **j**.

* This gives us the index of required element exist in the physical **1D** array.

EXAMPLE (1).

$i = 1$

$j = 2$

Find address of this **2D** in **1D**?

index of 1D

Address (index(1,2)) = BA + (1 x 3 + 2) x ES

↳ no of columns which we want to skip.

Address (index(1,2)) = 100 + (3+2) x 2

= 100 + 5 x 2

= 100 + 10

= 110

EXAMPLE (2)

$i = 2$

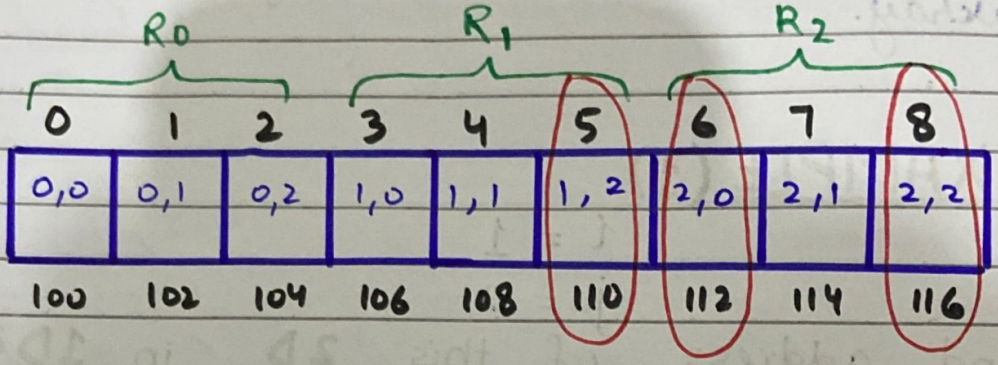
$j = 0$

Address(A[2][0]) = 100 + (2x3+0) x 2 = 112

EXAMPLE (3)

$i = 2$
$j = 2$

Address (A[2][2]) = 100 + (2×3 + 2) × 2 = 116



CALCULATING ADDRESS OF MEMORY FOR 3-D ARRAY

(ROW MAJOR ORDER MAPPING FORMULA)

$A[u_1][u_2][u_3]$

★ To access any arbitrary element i, j, K following mapping can be used

$$A(i,j,K) = BA + (iu_2u_3 + ju_3 + K) \times ES$$

↳ By this, we get mapping of 3D in 1D (Address)

CALCULATING ADDRESS OF MEMORY FOR 4-D ARRAY

(ROW MAJOR ORDER MAPPING FORMULA)

A[u_1][u_2][u_3][u_4]

★ To access any arbitrary element i, j, K, l following mapping formula can be used:

$$A(i,j,K,l) = BA + (i u_2 u_3 u_4 + j u_3 u_4 + K u_4 + l) \times ES$$

GENERALIZED FORMULA (ROW MAJOR ORDER MAPPING FORMULA)

$$A[u_1][u_2][u_3]\dots[u_{n-1}][u_n]$$

★ To access any arbitrary element i, j, K, L, ..., Z, following formula can be used:

$$\text{Address}(A[i][j]\dots[y][z])$$

$$= BA + (iu_2u_3\dots u_n + ju_3\dots u_n + \dots + yu_n + 1z) \times ES$$