1. *Memory Representation of Arrays*

An array allocates contiguous memory locations. Each element occupies memory equal to the size of its data type. If the data type is int and one integer occupies 4 bytes, then each element of the array occupies 4 bytes.

In Single Dimensional Array:

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

4 bytes	4bytes	4bytes	4bytes	4bytes
$elem = 1$	$elem = 2$	$elem = 3$	$elem = 4$	$elem = 5$
0	1	2	3	4

$$arr[0] = 1$$

$$arr[1] = 2$$

$$arr[2] = 3$$

$$arr[3] = 4$$

$$arr[4] = 5$$

Here elements $= 1, 2, 3, 4, 5$ *and index* $= 0, 1, 2, 3, 4.$

And each element of 4 *bytes each stored in* 5 *indexes*

$[\ 5$ *contiguous memory locations*$] = 5 \times 4$ *bytes*

$=$ 20 *bytes in total. Means: The element stored in each indexes occupies* 4 *bytes.*

Two-Dimensional Array:

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

where ,row $= 3$ *and column* $= 4.$

$elem = 1$ [0][0]{4 bytes}	$elem = 2$ [0][1]{4 bytes}	$elem = 3$ [0][2]{4bytes}	$elem = 4$ [0][3]{4bytes}
$elem = 5$ [1][0]{4bytes}	$elem = 6$ [1][1]{4 bytes}	$elem = 7$ [1][2]{4 bytes}	$elem = 8$ [1][3]{4 bytes}
$elem = 9$ [2][0]{4 bytes}	$elem = 10$ [2][1]{4 bytes}	$elem = 11$ [2][2]{4 bytes}	$elem = 12$ [2][3]{4 bytes}

$$\boldsymbol{Hence\ Total\ Memory\ taken\ by\ the\ Array}$$

$$= \mathbf{4} \times \mathbf{3} \times \mathbf{4}\boldsymbol{bytes} = \mathbf{48}\ \boldsymbol{bytes}.$$

Memory Storage Order

In C/C++, multi-dimensional arrays are stored in row-major order.

This means elements are stored row by row in contiguous memory:

Row 0 → Row 1 → Row 2

Memory layout (conceptually):

[0][0], [0][1], [0][2], [0][3], [1][0], [1][1], [1][2], [1][3], [2][0], [2][1], [2][2], [2][3]

Three-Dimensional Array:

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

$$where\ , page\ size = 2, row\ = 3\ and\ column\ = 4.$$

Page 1

$elem = 1$ $[0][0][0]\{4\ bytes\}$	$elem = 2$ $[0][0][1]\{4\ bytes\}$	$elem = 3$ $[0][0][2]\{4bytes\}$	$elem = 4$ $[0][0][3]\{4bytes\}$
$elem = 5$ $[0][1][0]\{4bytes\}$	$elem = 6$ $[0][1][1]\{4\ bytes\}$	$elem = 7$ $[0][1][2]\{4\ bytes\}$	$elem = 8$ $[0][1][3]\{4\ bytes\}$
$elem = 9$ $[0][2][0]\{4\ bytes\}$	$elem = 10$ $[0][2][1]\{4\ bytes\}$	$elem = 11$ $[0][2][2]\{4\ bytes\}$	$elem = 12$ $[0][2][3]\{4\ bytes\}$

Page 2

$elem = 13$ $[1][0][0]\{4\ bytes\}$	$elem = 14$ $[1][0][1]\{4\ bytes\}$	$elem = 15$ $[1][0][2]\{4bytes\}$	$elem = 16$ $[1][0][3]\{4bytes\}$
$elem = 17$ $[1][1][0]\{4bytes\}$	$elem = 18$ $[1][1][1]\{4\ bytes\}$	$elem = 19$ $[1][1][2]\{4\ bytes\}$	$elem = 20$ $[1][1][3]\{4\ bytes\}$
$elem = 21$ $[1][2][0]\{4\ bytes\}$	$elem = 22$ $[1][2][1]\{4\ bytes\}$	$elem = 23$ $[1][2][2]\{4\ bytes\}$	$elem = 24$ $[1][2][3]\{4\ bytes\}$

Total memory it takes: $2 \times 3 \times 4 \times 4bytes = 96\ bytes$.

Memory Storage Order (Row-Major in 3D)

C/C++ stores 3D arrays in row-major order as well.

Storage sequence:

[0][0][0], [0][0][1], [0][0][2], [0][0][3], [0][1][0], [0][1][1], [0][1][2], [0][1][3], [0][2][0], [0][2][1], [0][2][2], [0][2][3], [1][0][0], [1][0][1], ... , [1][2][3]

Thus, elements are stored continuously in memory.

***Note**: As it takes adjacent contiguous memory location it looks like this*: