4.5.POINTERS AND MULTI-DIMENSIONAL ARRAYS

If a one-dimensional array is like a list of elements, a two-dimensional array is like a table or a matrix.The elements in a 2D array can be considered to be logically arranged in rows and columns. Hence, the location of any element is decided by two indices, its row number and column number. Both row and column indexes start from "0".

```
int arr[2][2];
```

Such an array is represented as −

	Col0	**Col1**	**Col2**
Row0	arr[0][0]	arr[0][1]	arr[0][2]
Row1	arr[1][0]	arr[1][1]	arr[1][2]
Row2	arr[2][0]	arr[2][1]	arr[2][2]

It may be noted that the tabular arrangement is only a logical representation. The compiler allocates a block of continuous bytes. In C, the array allocation is done in a row-major manner, which means the elements are read into the array in a row−wise manner.

Here, we declare a 2D array with three rows and four columns (the number in the first square bracket always refers to the number of rows) as −

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

The compiler will allocate the memory for the above 2D array in a row−wise order. Assuming

that the first element of the array is at the address 1000 and the size of type "int" is 4 bytes, the elements of the array will get the following allocated memory locations −

	Row 0				**Row 1**				**Row 2**			
Value	1	2	3	4	5	6	7	8	9	10	11	12
Address	1000	1004	1008	1012	1016	1020	1024	1028	1032	1036	1040	1044

We will assign the address of the first element of the array num to the pointer ptr using the address of & operator.

```
int *ptr = &arr[0][0];
```

Example 1

If the pointer is incremented by 1, it moves to the next address. All the 12 elements in the "3×4" array can be accessed in a loop as follows −

```
#include <stdio.h>
int main(){
  int arr[3][4] = {
    {1, 2,  3,  4},
    {5, 6,  7,  8},
    {9, 10, 11, 12},
  };
   int *ptr = &arr[0][0];
  int i, j, k = 0;
  for (i = 0; i < 3; i++)
    { for (j = 0; j < 4; j+
    +){
      printf("%d  ", *(ptr + k));
      k++;
    }
    printf("\n");
  }
    return 0;
```

Output

When you run this code, it will produce the following output −

1 2 3 4

5 6 7 8

9 10 11 12

In general, the address of any element of the array by with the use the following formula −

```
add of element at ith row and jth col = baseAddress + [(i * no_of_cols + j) * sizeof(array_type)]
```

In our 3×4 array,

```
add of arr[2][4] = 1000 + (2*4 + 2)*4 = 1044
```

You can refer to the above figure and it confirms that the address of "arr[3][4]" is 1044.