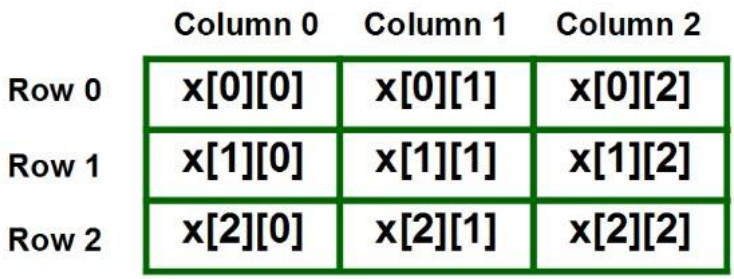
Matrix Implementation

Matrix in programming languages can be implemented using **2-D arrays**. 2-D arrays or Two-Dimensional arrays in simple words can be defined as an *array of arrays*.

Elements in a 2-D array are stored in a tabular form in row major order. A two – dimensional array can be seen as a table with 'x' rows and 'y' columns where the row number ranges from 0 to (x-1) and column number ranges from 0 to (y-1). A two – dimensional array with 3 rows and 3 columns is shown below:



Declaring 2-D Arrays:

- Syntax for declaring 2-D Array in C++:

```
data_type  array_name[size1][size2]

Where,
data_type: Type of data to be stored in the array.
           Here data_type is valid C/C++ data type
array_name: Name of the array
size1: Number of rows
size2: Number of columns

Example:
int arr[2][5];

The above example, creates a 2-D array named arr with
2 rows and 5 columns in C/C++.
```

- Declaring 2-D array in Java:

```
data_type[][] array_name = new data_type[size1][size2]

Where,
data_type: Type of data to be stored in the array.
           Here data_type is valid Java data type
array_name: Name of the array
size1: Number of rows
size2: Number of columns

Example:
int arr[2][5];

The above example, creates a 2-D array named arr with
2 rows and 5 columns in Java.
```

Size of 2-D arrays: The total number of elements that can be stored in a 2-D array can be easily calculated by multiplying the size of both dimensions. For Example, the above declared array *arr* can store a maximum of 2*5 = 10 elements.

Accessing 2-D array elements

Elements in two-dimensional arrays are commonly referred by x[i][j] where 'i' is the row number and 'j' is the column number.

Syntax:

```
arr[row_index][column_index]
```

For example:

```
arr[0][0] = 1;
```

The above example represents the element present in first row and first column.

Note: In arrays if size of array is N. Its index will be from 0 to N-1. Therefore, for row_index 2, actual row number is 2+1 = 3.

Printing all elements of a 2-D array: To print all the elements of a Two-Dimensional array we can use nested for loops. We will require two for loops. One to traverse the rows and another to traverse columns.

Consider a 2-D array named **arr[][]** has **N** rows and **M** columns. Below code snippet traverses all of the elements of the 2-D array in row-major order and prints them:

```
// Traversing number of Rows
for (int i = 0; i < N; i++)
{
    // Traversing number of Columns
    for (int j = 0; j < M; j++)
    {
        // Access each element and print it
        print arr[i][j];
    }
}
```

Searching an element in a 2-D array: We can use a similar approach as above to search a given element if it is present in a 2-D array arr[] or not. The idea is to traverse the 2-D array using two nested loops and check for every element of the 2-D array if it matches with the given element. We will use a boolean flag, which will be set to true if the element is found in the 2-D array.

Consider a 2-D array named **arr[][]** has **N** rows and **M** columns. Below code snippet traverses all of the elements of the 2-D array in row-major order and check if the element **key** exists in it or not:

```
// Declare a boolean flag variable,initialized to false
boolean flag = false;
// Traversing number of Rows
for (int i = 0; i < N; i++)
{
    // Traversing number of Columns
    for (int j = 0; j < M; j++)
    {
        // Check if key is present Set flag to true and stop  traversing further
        if(arr[i][j] == key)
        {
            flag = true;
            break;
        }
    }
    // If element found in the current row,stop traversing further
    if(flag == true)
        break;
}
// The flag is now True if the element is present in
// the array otherwise it will be false.
```