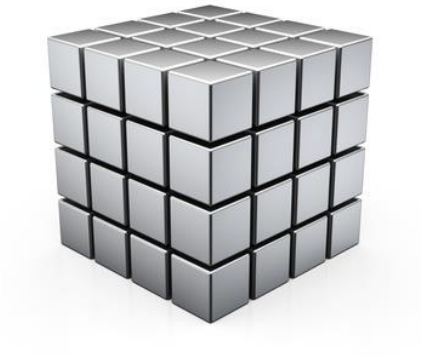
2-Dimensional Arrays



In most programming languages you can create **multi-dimensional** arrays. Multidimensional arrays are necessary to:

- Perform various mathematical operations (matrices)
- Model 2-dimentioanal and 3 dimensional space
- Model data tables.

Here we will look at **2-Dimensional** arrays and how they can be used.

What is a 2D dimensional array? Look at the image to the right. 2D arrays are simply a 2-dimentions grid of data. Storing data this way can make things even easier to work with and helps us perform more complex tasks.

	Columns		
Rows	score [0] [0]	score [0] [1]	score[0] [2]
	score[1] [0]	score[1] [1]	score[1] [2]
	score[2] [0]	score[2] [1]	score[2] [2]

Fig: Two-dimensional array

A two-dimensional array is, in essence, a list of one-dimensional arrays.

How to create (declare) a 2-D array:

3 rows, 4 columns:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
int a[3][4] =  {

  {0, 1, 2, 3},   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7},   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */

        };   notice the enclosing brackets at start and end of declaration
```

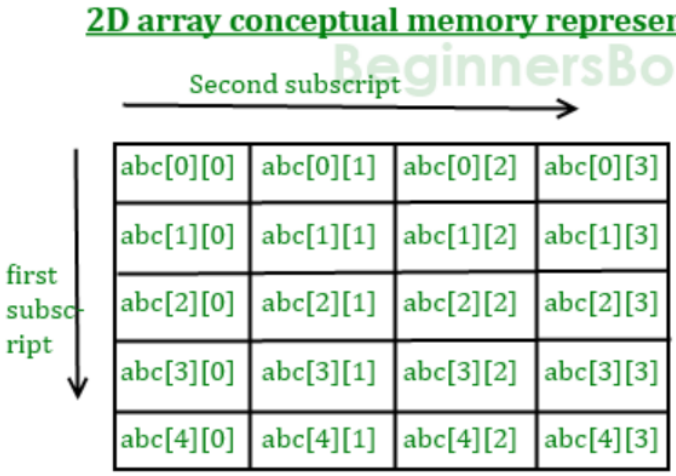
IMPORTANT!...Notice how that when you declare the [3][4] array **it's locations (index) are still named starting with zero**. (0,1,2) are the names for the rows in a 3 row array.

The 3rd row in the 4th column has a location of `a[2][3]` tricky!

We could also declare the array above in the following way:

int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11}; #**4** would be stored in the second row, first column with the address a[1][0].

In the Example of a [5][4] array below **NOTICE the location address of the 5th row 4th column is `abc[4][3]`**



Exercise 6.1 (make a prediction then **copy and paste**)

Look at the **array** declaration below. What would the array look like? **Where would the numbers in the list go** in the array? Now, try to figure out what the following code will output. Then put it into the compiler and see what it does.

```
#include <stdio.h>
int main ()
{
   /* an array with 2 rows and 3 columns*/
int table[2][3] = {10, 22, 33, 44, 45, 78 };

 int i, j;

   /* output each array element's value */
   for ( i = 0; i < 2; i++ )
       {
          for ( j = 0; j < 3; j++ )
          {
          printf("table[%d][%d] = %d\t", i,j, table[i][j] );
          }
          printf("\n");  /*after printing first row, move to next*/
       }

   return 0;
}
```

NOTICE:

"*Nested*" `for` loops (One inside of another)

Most common way to work with 2D arrays.

Make sure you know how this works

How to store user input data into 2D array:

We can **store** the elements in a 2-D array in a way similar to the method used above. Using **two "for" loops** (one nested within the other).

Example:

```
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int abc[3][4];
   /*Counter variables for the loop*/
   int i, j;
   for(i=0; i<3; i++)
      {
      for(j=0;j<4;j++)
         {
         printf("Enter value for abc[%d][%d]:", i, j);
         scanf("%d", &abc[i][j]);
         }
       }
   return 0;
}
```

Exercise 6.2 (You do on your own)

Run the code above in the online compiler to see how it works. Then combine this code with the code in exercise 6.1 to create program that gets the user to fill up an array and then it gets displayed (correctly) on the screen. The array should be displayed with the correct number of rows and columns.

Matrices:

In mathematics a collection of data in rows and columns is called a **Matrix.** Matrices are incredibly useful things that crop up in many different applied areas. Software is often used to manipulate large matrices. For now we will do one elementary operation with matrices, but you should not be surprised to encounter matrices again in, say, physics, engineering, or business.

Matrix Addition

Matrix addition is done by simply adding corresponding values in one matrix with the corresponding values in another matrix i.e. Sum of two matrices **A** and **B** is defined by:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+9 & 2+8 & 3+7 \\ 4+6 & 5+5 & 6+4 \\ 7+3 & 8+2 & 9+1 \end{bmatrix}$$

$$= \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix}$$

Exercise 6.3

*(**try on your own** using code from 6.1 and 6.2 then peek at the solution)*

Create a program that reads two separate matrices and adds them like the example above.
Example Input/Output:

```
Enter elements in matrix A of size 3x3:
1 2 3
4 5 6
7 8 9

Enter elements in matrix B of size 3x3:
9 8 7
6 5 4
3 2 1

Sum of matrices A+B =
10 10 10
10 10 10
10 10 10
```

Sample solution:

```c
#include <stdio.h>
#define SIZE 3 // Size of the matrix
int main()
{
    int A[SIZE][SIZE]; // Matrix 1
    int B[SIZE][SIZE]; // Matrix 2
    int C[SIZE][SIZE]; // Resultant matrix
    int row, col;
    /* Input elements in first matrix*/
    printf("Enter elements in matrix A of size 3x3: \n");
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            scanf("%d", &A[row][col]);
        }
    }
    /* Input elements in second matrix */
    printf("\nEnter elements in matrix B of size 3x3: \n");
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            scanf("%d", &B[row][col]);
        }
    }
       /* Add A and B array elements and store the sums in array C */
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
          C[row][col] = A[row][col] + B[row][col];
        }
    }
    /* Print the value of resultant matrix C */
    printf("\nSum of matrices A+B = \n");
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            printf("%d ", C[row][col]);
        }
        printf("\n");
    }
    return 0;}
```

Exercise. 6.4 (tough one – but try on your own, then check)

Write a program to find sum of each row and the sum of each column of an array.

Sample Input:

```
Input the size of a square matrix : 2   {meaning 2 x 2}
Input elements in the matrix :
element - [0],[0] : 5
element - [0],[1] : 6
element - [1],[0] : 7
element - [1],[0] : 8
```

***Sample Output*:**

```
The matrix is :
5 6
7 8
The sum or rows and columns of the matrix is :
5 6 11
7 8 15

12 14
```

Sample solution:

```
#include <stdio.h>

void main()
{
     int i,j,k,arr1[10][10],rsum[10],csum[10],n;
       printf("\n\nFind the sum of rows an columns of a Matrix:\n");
       printf("-----------------------------------------\n");

     printf("Input the size of the square matrix : ");
     scanf("%d", &n);
       printf("Input elements in the first matrix :\n");
       for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                 printf("element - [%d],[%d] : ",i,j);
                 scanf("%d",&arr1[i][j]);
            }
        }

/* more code on next page */
```

```
printf("The matrix is :\n");
       for(i=0;i<n;i++)
       {
         for(j=0;j<n ;j++)
           printf("% 4d",arr1[i][j]);
          printf("\n");
       }

     /* Sum of rows */
     for(i=0;i<n;i++)
     {
        rsum[i]=0;
        for(j=0;j<n;j++)
        rsum[i]=rsum[i]+arr1[i][j];
     }

      /* Sum of Column */
      for(i=0;i<n;i++)
      {
        csum[i]=0;
        for(j=0;j<n;j++)
            csum[i]=csum[i]+arr1[j][i];
      }

      printf("The sum or rows and columns of the matrix is :\n");
      for(i=0;i<n;i++)
      {
         for(j=0;j<n;j++)
            printf("% 4d",arr1[i][j]);
         printf("% 8d",rsum[i]);
         printf("\n");
       }
       printf("\n");
           for(j=0;j<n;j++)
              {
               printf("% 4d",csum[j]);
              }
            printf("\n\n");
  }
```