ARRAYS

Two Dimension Arrays (2D - Array)

- It is the simplest form of multi-dimensional arrays.
- Since in the 2D arrays the elements are arranged in rows and columns, generally they are known as matrix / elements arranged in table format.
- Two-dimensional (2D) arrays are indexed by two subscripts, one for the ***row*** and one for the ***column***.

Declaration of 2D arrays:

- Syntax:**Data_type Array_name[row_size][column_size];**

    - **Data_type** –general data types like - int, char, float, double.
    - **Array_name** – valid identifier.
    - **row_size**- indicates the subscript value for the maximum number of rows.
    - **column_size**- indicates the subscript value for the maximum number of columns.

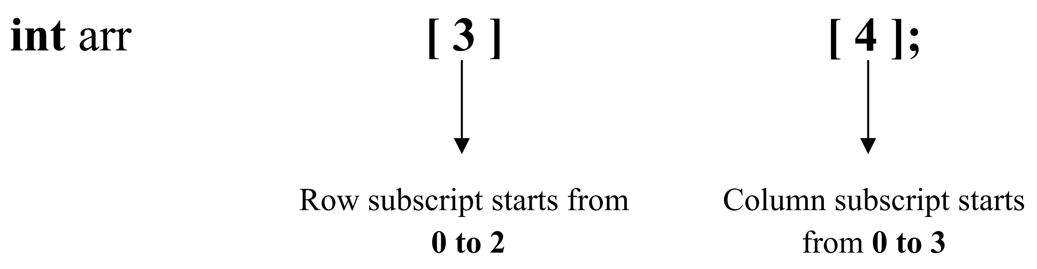
Example:intarr[3][4];

- In the above declaration the data type is integer
- The name of the 2D array is "arr"
- The maximum number of rows are 3
- The maximum number of columns are 4
- The maximum numbers of elements can be stored is 3*4=12 elements.

Pictorial representation of 2D – Array-

For the above declaration pictorial representation is shown below:

Column index → Row index ↓	0	1	2	3
0	**arr[0][0]**	**arr[0][1]**	**arr[0][2]**	**arr[0][3]**
1	**arr[1][0]**	**arr[1][1]**	**arr[1][2]**	**arr[1][3]**
2	**arr[2][0]**	**arr[2][1]**	**arr[2][2]**	**arr[2][3]**



int arr [3] [4];

Row subscript starts from **0 to 2**

Column subscript starts from **0 to 3**

Note:

✓ The memory allocated to the two dimensional array can be calculated as

Memory allocation= Size of the data type * Row size * column size

Ex: float arr[2][3];

✓ Memory allocated= 4 bytes (Size of float)* 2(Row Size) * 3 (Column Size)

Therefore total Memory allocated=24 bytes

✓ The 2D array elements are represented as follows:

Ex:int arr[3][4];

$$\begin{pmatrix} 2 & 4 & 1 & 10 \\ 6 & 7 & 5 & 3 \\ 8 & 11 & 14 & 20 \end{pmatrix}_{3X4}$$

✓ Elements of the 2D array is represented using array name with Subscript / Index as follows:

Representation	**Array Element**
a[0][0]	2
a[0][1]	4
a[0][2]	1
a[0][3]	10
a[1][0]	6
a[1][1]	7
a[1][2]	5
a[1][3]	3
a[2][0]	8
a[2][1]	11
a[2][2]	14
a[2][3]	20

Initialization of 2D arrays

✓ The different types of initializing arrays:

1. At Compile time

(i) Complete array initialization

(ii) Partial array initialization

.

2. At Run Time

✓ 2D arrays may be initialized by specifying bracketed values for each row.

Compile Time Initialization

- Assigning / providing the required value to the variable during declaration is termed as initialization.
- We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
- During Compile time - Arrays can be initialized at the time of declaration when their initial values are known in advance.
- The general form of initialization of 2D arrays is:

Data_type array-name [row_size][column size]=

{ list of values to be assigned row wise with in brackets};

Complete 2D array Initialization:

✓ In this kind of initialization all the memory locations will be assigned with a value during declaration itself.

Syntax:

Data_typeArray_name [Exp1][Exp2]={
{a_1,a_2,a_3….a_n},
{b_1,b_2,b_3….b_n},
……………...
{z_1,z_2,z_3…….z_n}
};

Where,

Data_type → may be general data type like int, char, float, double.

Array_name → is a valid identifier.

Exp1 → indicates the subscript value for the maximum number of rows.

Exp2 →indicates the subscript value for the maximum number of columns.

a_1,a_2,a_3….a_n → values assigned to 1st row

b_1,b_2,b_3….b_n → values assigned to 2nd row

z_1,z_2,z_3….z_n → values assigned to last row

Example:

```
int arr[3][4]=  {
                        { 13,45,63,89},
                        { 10,01,90,21},
                        { 34,07,81,23}
                };
```

- The pictorial representation of the above initialization is shown below

Column index →	0	1	2	3
Row index ↓				
0	**13**	**45**	**63**	**89**
1	**10**	**1**	**90**	**21**
2	**34**	**7**	**81**	**23**

Note:

- ✓ There are many ways to initialize two Dimensional arrays, the another way of complete initialization is shown below.

  int arr[3][4]= { 13,45,63,89, 10,1,90,21, 34,7,81,23};

  The representation is same as the above shown diagram.

Partial Initialization of 2D Array:

- ✓ In this kind of initialization the numbers of values to be initialized are less than the size of the array.
- ✓ The elements are initialized from left to right one after the other and the remaining locations are set to zero automatically.

Example:

```
int arr[3][4]= {
            { 13,45,63},
            { 10,1},
            { 34,7,81}
            };
```

- The pictorial representation of the above initialization is shown below

Column index →	0	1	2	3
Row index ↓				
0	**13**	**45**	**63**	**0**
1	**10**	**1**	**0**	**0**
2	**34**	**7**	**81**	**0**

Note:

- ✓ But when we go for the alternative way of initialization the representation will be different, the elements are initialized from left to right and row wise not column wise.

Example:

intarr[3][4]= { 13,45,63,89, 10,1,90,21, 34 };

Column index →	0	1	2	3
Row index ↓				
0	**13**	**45**	**63**	**89**
1	**10**	**1**	**90**	**21**
2	**34**	**0**	**0**	**0**

Very Important

- Generally the single dimensional array can be initialized without size also, but When a 2D array is defined it id mustto specify the second dimension even if you are giving values during the declaration.

Example:

/* Valid declaration*/

int abc[2][2] = {1, 2, 3 ,4 }

/* Valid declaration*/

int abc[][2] = {1, 2, 3 ,4 }

/* Invalid declaration – you must specify second dimension*/

int abc[][] = {1, 2, 3 ,4 }

/* Invalid because of the same reason mentioned above*/

int abc[2][] = {1, 2, 3 ,4 }

Run Time Initialization

- ✓ In Run time initialization2D - array is initialized explicitly at run time (During execution via output screen).
- ✓ This approach is usually applied for initializing large arrays and when initially values for an array are unknown.
- ✓ scanf() can be used to initialize an array at run time.

Ex:

```
int x[2][2];
scanf("%d%d%d%d",&x[0][0],&x[0][1],&x[1][0],&x[1][1]);
```

- ✓ The above statements will initialize array elements with the values entered through the key board for specified locations of the array.

(Or)

- ✓ For the arrays with larger size,for( ) loop can be used for initialization of an array.

Reading of values to 2D array:

- ✓ 1D array, array has a single subscript for which we can make use of one for loop to read the elements.
- ✓ In 2D as two subscripts are there each for row and column, we need two for loops to read the element to the 2D array.

Example: intarr[3][2];

Column index → Row index ↓	0	1
0	**arr[0][0]**	**arr[0][1]**
1	**arr[1][0]**	**arr[1][1]**
2	**arr[2][0]**	**arr[2][1]**

```
scanf("%d",&arr[0][0]);
scanf("%d",&arr[0][1]);
scanf("%d",&arr[1][0]);
scanf("%d",&arr[1][1]);
scanf("%d",&arr[2][0]);
scanf("%d",&arr[2][1]);
```

```
for(i=0;i<3;i++)
{
        for(j=0;j<2;j++)
        {
        scanf("%d",&a[i][j]);
        }
}
```

- During reading of array elements as we enter the array elements row wise initially the row value is fixed and column value changes, after completely enfettering elements to row, the next row starts reading the elements.
- So we use nested loops to read the elements for the 2D array, where outer loop will be for the number of rows(row count) and the inner loop is for the number of columns(column count).
- Generally if we consider two variables m(number of rows) and n(number of columns) , we can rewrite the statements for reading for loop as

```
for(i=0;i<m;i++)
{
         for(j=0;j<n;j++)
        {
        scanf("%d",&a[i][j]);
        }
}
```

Printingof values from 2D array

- As we came to know to read elements for 2D array we use nested for loop, similarly for printing also we make use of nested loops.

Example:intarr[3][2];

Column index → Row index ↓	0	1
0	**arr[0][0]**	**arr[0][1]**
1	**arr[1][0]**	**arr[1][1]**
2	**arr[2][0]**	**arr[2][1]**

- We can rewrite the statements for printing 2D elements using for loop as

```
for(i=0;i<m;i++)
{
         for(j=0;j<n;j++)
        {
        printf("%d\t",a[i][j]);
        }
printf("\n");
}
```

Write a C- Program to read and print the 2D array elements in terms of matrix.

```
#include<stdio.h>
#include<conio.h>

void main()
{
inti,j,m,n,arr[10][10];
clrscr();
     printf(“ enter the row and column size”);
     scanf(“%d%d”,&m,&n);
     printf(“\n enter the array elements\n”);
     for(i=0;i<m;i++)
     {
     for(j=0;j<n;j++)
     {
     scanf(“%d”,&a[i][j]);
     }
     }
     printf(“\n the entered array elements are\n”);
     for(i=0;i<m;i++)
     {
     for(j=0;j<n;j++)
     {
     printf(“%d\t”,a[i][j]);
     }
     printf(“\n”);
     }
     getch();
}
```

Exercise Programs:

1. **Write a C- Program to read a2D - array with m * n elements and to print the same.**
2. **Write a C- Program to copy one 2D – array in to another 2D – array.**
3. **Write a C- Program to find the sum of all the elements in 2D – array with m*n size.**
4. **Write a C- Program to find the largest element in the 2D - array.**
5. **Write a C- Program perform the matrix Addition.**
6. **Write a C- Program perform the matrix Subtraction.**
7. **Write a C- Program perform the matrix Multiplication.**
8. **Write a C- Program to find the row sum and column sum of the matrix.**
9. **Write a C- Program to find the trace (sum of principle diagonal elements) of the matrix.**
10. **Write a C- Program to find the Norm of the matrix.**
11. **Write a C- Program to find the Transpose of the matrix.**
12. **Write a C- Program to check a given matrix is Scalar matrix or not.**