

MULTI-DIMENSIONAL ARRAYS IN C

MULTIDIMENSIONAL ARRAY DECLARATION

- C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration

```
type name[size1][size2]...[sizeN];
```

```
char vowels[1][5] = { {'a', 'e', 'i', 'o', 'u'} };
```

```
char vowels[][5] = { {'A', 'E', 'I', 'O', 'U'},  
                      {'a', 'e', 'i', 'o', 'u'} };
```

- For example, the following declaration creates a three dimensional integer array

```
int threedim[5][10][4];
```



TWO-DIMENSIONAL ARRAYS

- To declare a two-dimensional integer array of size [x][y],

type arrayName [x][y];

Where

type can be any C data type (int, char, long, long long, double, etc.)

arrayName will be a valid C identifier, or variable.

- A two-dimensional array can be considered as a table which will have [x] number of rows and [y] number of columns.
- A two-dimensional array a, which contains three rows and four columns.
- Every element in the array a is identified by an element name in the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the indexes that uniquely identify, or show, each element in 'a'.

	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]

SYNTAX OF A MULTI DIMENSIONAL ARRAY IN C


PROGRAMMING

Data_Type Array_Name[Tables][Row_Size][Column_Size]

- **Data_type:** It will decide the type of elements it will accept. For example, If we want to store integer values then we declare the Data Type as int, If we want to store Float values then we declare the Data Type as float etc
- **Array_Name:** This is the name you want to give it to Multi Dimensional array in C.
- **Tables:** It will decide the number of tables an array can accept. Two Dimensional Array is always a single table with rows and columns. In contrast, Multi Dimensional array in C is more than 1 table with rows and columns.
- **Row_Size:** Number of Row elements an array can store.
- **Column_Size:** Number of Column elements an array can store.
- We can calculate the maximum number of elements in a Three Dimensional using: [Tables] * [Row_Size] * [Column_Size].

EXAMPLE

int Employees[2][4][3];

- Here, we used int as the data type to declare an array. So, the above array will accept only integers. If you try to add float values, it throws an error.
 - Employees is the array name
 - The number of tables = 2. So, this array will hold a maximum of 2 levels of data (rows and columns).
 - The Row size of an Array is 4. It means Employees array only accept 4 integer values as rows.
 - If we try to store more than 4, it throws an error.
 - We can store less than 4. For Example, If we store 2 integer values, the remaining two will assign with the default value (Which is 0).
 - The Column size of an Array is 3. It means Employees array will only accept 3 integer values as columns.
 - If we try to store more than 3, it throws an error.
 - We can store less than 3. For Example, If we store 1 integer values, the remaining 2 values will assign with the default value (Which is 0).
 - Finally, Employees array can hold a maximum of 24 integer values ($2 * 4 * 3 = 24$).
- 

INITIALIZING A MULTIDIMENSIONAL ARRAY

- Initialization of a 2d array

- // Different ways to initialize two-dimensional array

```
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};
```

```
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
```

```
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

- Initialization of a 3d array

- You can initialize a three-dimensional array in a similar way like a two-dimensional array.

```
int test[2][3][4] =
```

```
{ {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},
```

```
{{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```



INITIALIZING TWO-DIMENSIONAL ARRAYS

- Multidimensional arrays may be used by specifying bracketed[] values for each row.

- Below is an array with 3 rows and each row has 4 columns.

```
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 */  
               };
```

- The inside braces, which indicates the wanted row, are optional.

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```



INITIALIZING TWO-DIMENSIONAL ARRAYS

- A two dimensional array:

int arr[2][3];

- This array has total $2*3 = 6$ elements.

- A three dimensional array:

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

- This array has total $2*2*2 = 8$ elements.



INITIALIZATION OF 2D ARRAY

```
1 int temp[2][3] = {  
2     { 1, 2, 3 }, // row 0  
3     {11, 22, 33} // row 1  
4 };
```

int temp[2][3] = {
row 0 → { 1, 2, 3 },
row 1 → {11, 22, 33},
};
col 0 col 1 col 2

```
temp[0][0] : 1  
temp[0][1] : 2  
temp[0][2] : 3  
temp[1][0] : 11  
temp[1][1] : 22  
temp[1][2] : 33
```

INITIALIZATION OF 2D ARRAY

- There are two ways to initialize a two Dimensional arrays during declaration.

- `int disp[2][4] =`

```
{  
  {10, 11, 12, 13},  
  {14, 15, 16, 17}  
};
```

- `int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};`

- `/* 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 }
```



Array declaration, initialization and accessing

Example

Array declaration syntax:

```
data_type arr_name  
[num_of_rows][num_of_column];
```

Array initialization syntax:

```
data_type arr_name[2][2] =  
{{0,0},{0,1},{1,0},{1,1}};
```

Array accessing syntax:

```
arr_name[index];
```

Integer array example:

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

```
arr [0] [0] = 1;  
arr [0] ]1] = 2;  
arr [1][0] = 3;  
arr [1] [1] = 4;
```