**Multidimensional Arrays in C**

In C, we can define multidimensional arrays in simple words as array of arrays. Data in multidimensional arrays are stored in tabular form (in row major order).

**Syntax :** General form of declaring N-dimensional arrays:

**data\_type array\_name[size1][size2]....[sizeN];**

Where

**data\_type**: Type of data to be stored in the array. Here **data\_type** is valid C data type

**array\_name**: Name of the array

**size1, size2,... ,sizeN**: Sizes of the dimensions

**Examples**:

int two\_d[10][20];

int three\_d[10][20][30];

**Size of multidimensional arrays**

Total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

**For example:**

The array **int x[10][20]** can store total (10\*20) = 200 elements.

Similarly array **int x[5][10][20]** can store total (5\*10\*20) = 1000 elements.

**Two-Dimensional Array**

* An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns.The basic form of declaring a two-dimensional array of size x, y:
* **Syntax:**

**data\_type array\_name[x][y];**

**where**

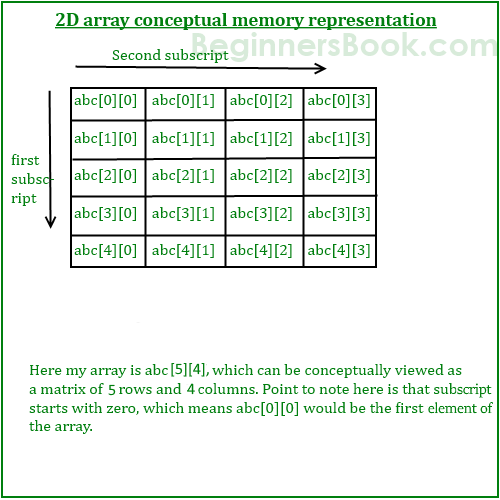
**data\_type**: Type of data to be stored. Valid C data type.

We can declare a two dimensional integer array say ‘x’ of size 10,20 as:

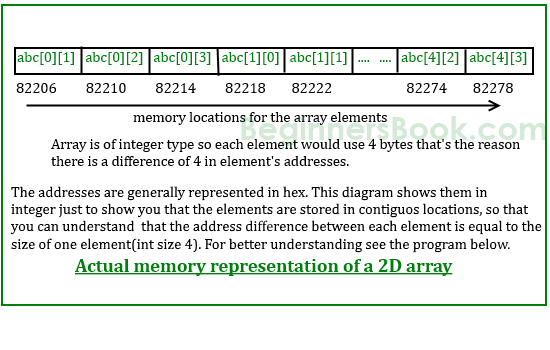
int x[10][20];

* Elements in two-dimensional arrays are commonly referred by x[i][j] where i is the row number and ‘j’ is the column number.
* 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 ‘x’ with 3 rows and 3 columns is shown below:

int abc[5][4] : there are 5 row and each row contain 4 column



However the actual representation of this array in memory would be something like this:



**Initializing Two – Dimensional Arrays**:

There are two ways in which a Two-Dimensional array can be initialized.  
**First Method**:

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

The above array have 3 rows and 4 columns. The elements in the braces from left to right are stored in the table also from left to right. The elements will be filled in the array in the order, first 4 elements from the left in first row, next 4 elements in second row and so on.

**Better Method**:

int x[3][4] = {

{0,1,2,3},

{4,5,6,7},

{8,9,10,11}

};

This type of initialization make use of nested braces. Each set of inner braces represents one row. In the above example there are total three rows so there are three sets of inner braces.

**Things that you must consider while initializing a 2D array**

We already know, when we initialize a normal array (or you can say one dimensional array) during declaration, we need not to specify the size of it. However that’s not the case with 2D array, you must always specify the second dimension even if you are specifying elements during the declaration. Let’s understand this with the help of few examples –

/\* 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 }

**Accessing Elements of Two-Dimensional Arrays:**

Elements in Two-Dimensional arrays are accessed using the row indexes and column indexes.  
Example:

**int x[2][1];**

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

#include<stdio.h>

int main(){

/\* 2D array declaration\*/

int abc[5][4];

/\*Counter variables for the loop\*/

int i, j;

for(i=0; i<5; i++) {

for(j=0;j<4;j++) {

printf("Enter value for abc[%d][%d]:", i, j);

scanf("%d", &abc[i][j]);

}

}

return 0;

}

**Two-dimensional array example in C**

#include<stdio.h>

int main(){

int i=0,j=0;

int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

//traversing 2D array

for(i=0;i<4;i++){

for(j=0;j<3;j++){

printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);

}//end of j

}//end of i

return 0;

}

**Output**

arr[0][0] = 1

arr[0][1] = 2

arr[0][2] = 3

arr[1][0] = 2

arr[1][1] = 3

arr[1][2] = 4

arr[2][0] = 3

arr[2][1] = 4

arr[2][2] = 5

arr[3][0] = 4

arr[3][1] = 5

arr[3][2] = 6

**2D array example: Storing elements in a matrix and printing it.**

#include <stdio.h>

void main ()

{

int arr[3][3],i,j;

for (i=0;i<3;i++)

{

for (j=0;j<3;j++)

{

printf("Enter a[%d][%d]: ",i,j);

scanf("%d",&arr[i][j]);

}

}

printf("\n printing the elements ....\n");

for(i=0;i<3;i++)

{

printf("\n");

for (j=0;j<3;j++)

{

printf("%d\t",arr[i][j]);

}

}

}

**Output**

Enter a[0][0]: 56

Enter a[0][1]: 10

Enter a[0][2]: 30

Enter a[1][0]: 34

Enter a[1][1]: 21

Enter a[1][2]: 34

Enter a[2][0]: 45

Enter a[2][1]: 56

Enter a[2][2]: 78

printing the elements ....

56 10 30

34 21 34

45 56 78

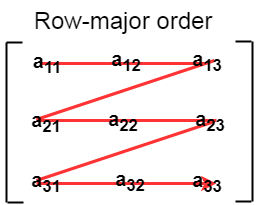
There are two main techniques of storing 2D array elements into memory

### 1. Row Major ordering

In row major ordering, all the rows of the 2D array are stored into the memory contiguously. Considering the array shown in the above image, its memory allocation according to row major order is shown as follows.

DS 2D Array

first, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.

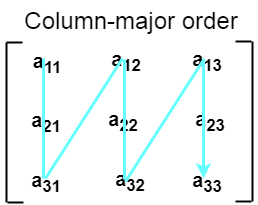


### 2. Column Major ordering

According to the column major ordering, all the columns of the 2D array are stored into the memory contiguously. The memory allocation of the array which is shown in in the above image is given as follows.

DS 2D Array

first, the 1st column of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last column of the array.



**Calculating the Address of the random element of a 2D array**

Due to the fact that, there are two different techniques of storing the two dimensional array into the memory, there are two different formulas to calculate the address of a random element of the 2D array.

**By Row Major Order**

If array is declared by a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,

**Address(a[i][j]) = B. A. + (i \* n + j) \* size**

where, B. A. is the base address or the address of the first element of the array a[0][0] .

**Example :** A matrix B[10][20] is stored in the memory with each element requiring 2 bytes of storage. If the base address is 0, find the address of B[5][4] when the matrix is stored in row Major Wise.

Address(a[5][4]) = 0 + ( 5\*20+4)\*2

= 0+(100+4)\*2

= 0+104\*2

= 208 answer

**By Column major order**

If array is declared by a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,

**Address(a[i][j]) = BA + ( I + ( j\*m ) ) \* Size**

where BA is the base address of the array.

**Example:** A matrix B[10][20] is stored in the memory with each element requiring 2 bytes of storage. If the base address is 0, find the address of B[5][4] when the matrix is stored in Column Major Wise.

Address [A[5][4]) = 0 + (5+(4\*10)) \* 2

= 0 + (5 + 40 ) \* 2

= 45 \* 2

= 90 byte

**Multidimentional Character Array in C**

**Definition of String ( Null Terminated Array ):** The array of characters is called a string. “Hi”, “Hello”, and e.t.c are the examples of String.

**Memory Representation of 1D character array:**

char name[5];

printf(“Enter your name:”);

gets(name); ----- > we enter “om”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| o | m | \0 |  |  |

The array of Strings is nothing but a two-dimensional (2D) array of characters.

**Declaration of 2D Array**

**char string-array-name[row-size][column-size];**

Here the first index (row-size) specifies the maximum number of strings in the array, and the second index (column-size) specifies the maximum length of every individual string.

**For example**:

char language[5][10];

In the “language” array we can store a maximum of 5 Strings and each String can have a maximum of 10 characters.

In C language, each character take 1 byte of memory. For the “language” array it will allocate 50 bytes (1\*5\*10) of memory. Where each String will have 10 bytes (1\*10) of memory space.

**Initialization of Array of string**

To declare an array of Strings in C, we must use the char data type.

**Example**:

char language [5][10] = {"Java", "Python", "C++", "HTML", "SQL"};

The two-dimensional (2D) array of Strings in C also can be initialized as,

char language[5][10] ={

{'J','a','v','a','\0'},

{'P','y','t','h','o','n','\0'},

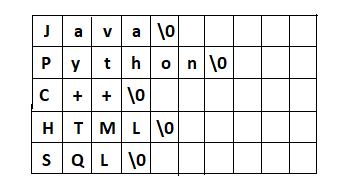
{'C','+','+','\0'},

{'H','T','M','L','\0'},

{'S','Q','L','\0'}

};

Since it is a two-dimension of characters, so each String (1-D array of characters) must end with null character i.e. ‘\0’



**Note 1:** the number of characters (column-size) must be declared at the time of the initialization of the two-dimensional array of strings.

// it is valid

char language[ ][10] = {"Java", "Python", "C++", "HTML", "SQL"};

But that the following declarations are invalid.

// invalid

char language[ ][ ] = {"Java", "Python", "C++", "HTML", "SQL"};

// invalid

char language[5][ ] = {"Java", "Python", "C++", "HTML", "SQL"};

**Note2:-** Once we initialize the array of String then we can’t directly assign a new String.

char language[5][10] = {"Java", "Python", "C++", "HTML", "SQL"};

// now, we can't directly assign a new String

language[0] = "Kotlin"; // invalid

// we must copy the String

strcpy(language[0], "Kotlin"); // valid

// Or,

scanf(language[0], "Kotlin"); // valid

**Reading and displaying 2d array of strings in C**

#include<stdio.h>

int main()

{

char name[10][20];

int i,n;

printf("Enter the number of names (<10): ");

scanf("%d",&n);

// reading string from user

printf("Enter %d names:\n",n);

for(i=0; i<n; i++)

scanf("%s[^\n]",name[i]);

// dispaying strings

printf("\nEntered names are:\n");

for(i=0;i<n;i++)

puts(name[i]);

return 0;

}