Multidimensional Arrays in C

A multi-dimensional array can be termed as an array of arrays that stores homogeneous data in tabular form. Data in multidimensional arrays is generally stored in row-major order in the memory.

The ***general form of declaring N-dimensional arrays*** is shown below.

Syntax:

```
data_type array_name[size1][size2]....[sizeN];
```

- **data_type**: Type of data to be stored in the array.
- **array_name**: Name of the array.
- **size1, size2,..., sizeN**: Size of each dimension.

Examples:

```
Two dimensional array: int two_d[10][20];

Three dimensional array: int three_d[10][20][30];
```

Size of Multidimensional Arrays:

The 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.

To get the size of the array in bytes, we multiply the size of a single element with the total number of elements in the array.

For example:

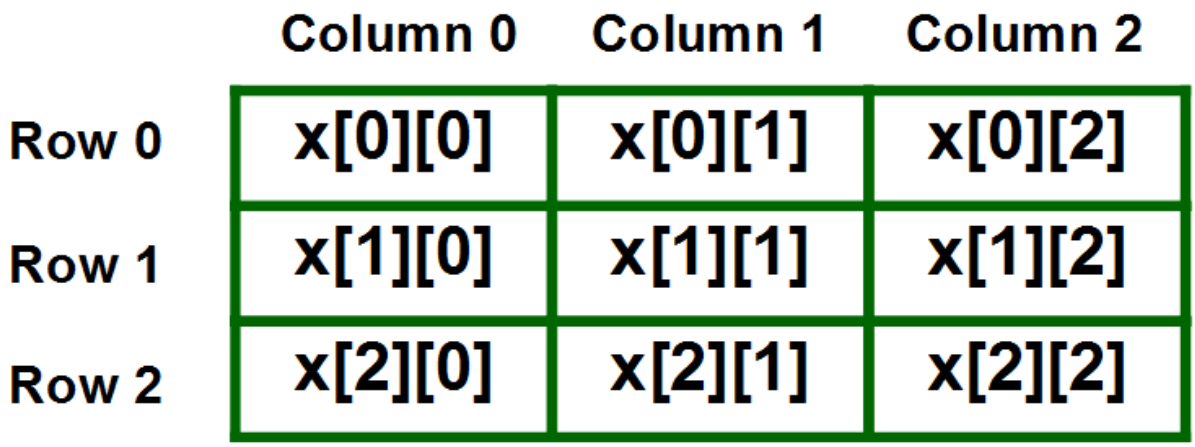
- Size of array **int x[10][20]** = 10 * 20 * 4 = 800 bytes. (where int = 4 bytes)
- Similarly, size of **int x[5][10][20]** = 5 * 10 * 20 * 4 = 4000 bytes. (where int = 4 bytes)

The most commonly used forms of the multidimensional array are:

1. **Two Dimensional Array**
2. **Three Dimensional Array**

Two-Dimensional Array in C

A **two-dimensional array** or **2D array** in C is the simplest form of the multidimensional array. We can visualize a two-dimensional array as an array of one-dimensional arrays arranged one over another forming a table with 'x' rows and 'y' columns where the row number ranges from 0 to (x-1) and the column number ranges from 0 to (y-1).



Graphical Representation of Two-Dimensional Array of Size 3 x 3

Declaration of Two-Dimensional Array in C

The basic form of declaring a 2D array with **x** rows and **y** columns in C is shown below.

Syntax:

```
data_type array_name[x][y];
```

where,

- **data_type:** Type of data to be stored in each element.
- **array_name:** name of the array
- **x:** Number of rows.
- **y:** Number of columns.

We can declare a two-dimensional integer array say 'x' with 10 rows and 20 columns as:

Example:

```
int x[10][20];
```

Note: In this type of declaration, the array is allocated memory in the stack and the size of the array should be known at the compile time i.e. size of the array is fixed. We can also create an array dynamically in C by using methods mentioned here.

Initialization of Two-Dimensional Arrays in C

The various ways in which a 2D array can be initialized are as follows:

1. **Using Initializer List**
2. **Using Loops**

1. Initialization of 2D array using Initializer List

We can initialize a 2D array in C by using an initializer list as shown in the example below.

First Method:

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

The above array has 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 order: the first 4 elements from the left will be filled in the first row, the next 4 elements in the second row, and so on.

Second Method (better):

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

This type of initialization makes use of nested braces. Each set of inner braces represents one row. In the above example, there is a total of three rows so there are three sets of inner braces. The advantage of this method is that it is easier to understand.

Note: The number of elements in initializer list should always be less than or equal to the total number of elements in the array.

2. Initialization of 2D array using Loops

We can use any C loop to initialize each member of a 2D array one by one as shown in the below example.

Example:

```
int x[3][4];


for(int i = 0; i < 3; i++){
    for(int j = 0; j < 4; j++){
        x[i][j] = i + j;
    }
}
```

This method is useful when the values of each element have some sequential relation.

Accessing Elements of Two-Dimensional Arrays in C

Elements in 2D arrays are accessed using row indexes and column indexes. Each element in a 2D array can be referred to by:

Syntax:

```
array_name[i][j]
```

where,

- **i:** The row index.
- **j:** The column index.

Example:

```
int x[2][1];
```

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

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

For printing the whole array, we access each element one by one using loops. The order of traversal can be row-major order or column-major order depending upon the requirement. The below example demonstrates the row-major traversal of a 2D array.

Example:

- C

```c
// C Program to print the elements of a
// Two-Dimensional array

#include <stdio.h>

int main(void)
{
    // an array with 3 rows and 2 columns.
    int x[3][2] = { { 0, 1 }, { 2, 3 }, { 4, 5 } };

    // output each array element's value
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            printf("Element at x[%i][%i]: ", i, j);
            printf("%d\n", x[i][j]);
        }
    }

    return (0);
}

// This code is contributed by sarajadhav12052009
```

Output

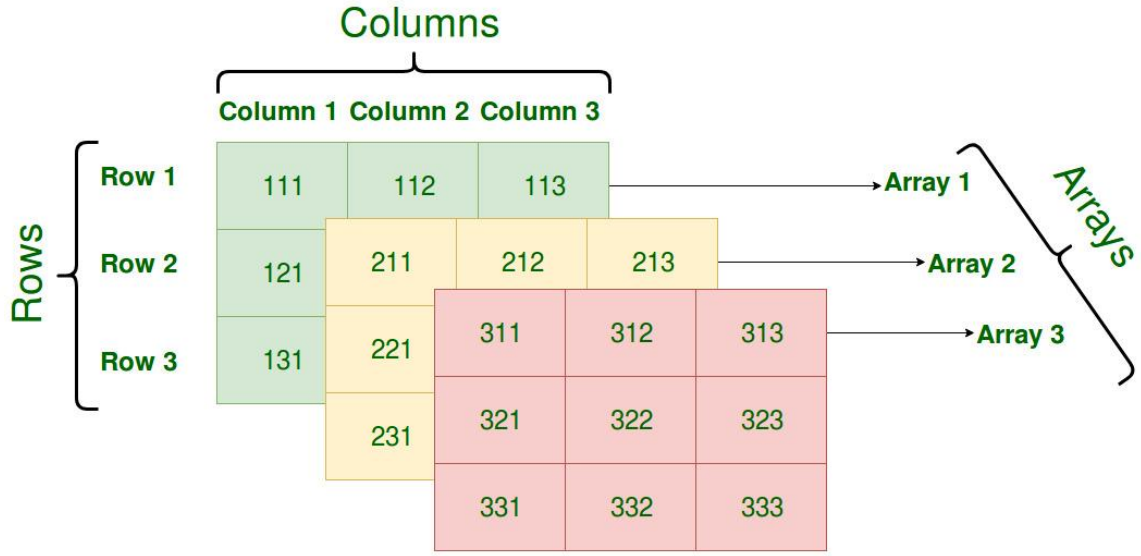
```
Element at x[0][0]: 0

Element at x[0][1]: 1

Element at x[1][0]: 2

Element at x[1][1]: 3

Element at x[2][0]: 4
```

```
Element at x[2][1]: 5
```

Time Complexity: O(N*M) ,where N(here 3) and M(here 2) are number of rows and columns repectively.
Space Complexity:O(1)

Three-Dimensional Array in C

A **Three Dimensional Array** or **3D** array in C is a collection of two-dimensional arrays. It can be visualized as multiple 2D arrays stacked on top of each other.



Graphical Representation of Three-Dimensional Array of Size 3 x 3 x 3

Declaration of Three-Dimensional Array in C

We can declare a 3D array with **x** 2D arrays each having **y** rows and **z** columns using the syntax shown below.
Syntax:

```
data_type array_name[x][y][z];
```

- **data_type:** Type of data to be stored in each element.
- **array_name:** name of the array
- **x:** Number of 2D arrays.
- **y:** Number of rows in each 2D array.
- **z:** Number of columns in each 2D array.

Example:

```
int array[3][3][3];
```

Initialization of Three-Dimensional Array in C

Initialization in a 3D array is the same as that of 2D arrays. The difference is as the number of dimensions increases so the number of nested braces will also increase.

A 3D array in C can be initialized by using:

1. **Initializer List**
2. **Loops**

Initialization of 3D Array using Initializer List

Method 1:

```
int x[2][3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                  11, 12, 13, 14, 15, 16, 17, 18, 19,
                  20, 21, 22, 23};
```

Method 2(Better):

```
int x[2][3][4] =
 {
   { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },
   { {12,13,14,15}, {16,17,18,19}, {20,21,22,23} }
 };
```

Initialization of 3D Array using Loops

It is also similar to that of 2D array with one more nested loop for accessing one more dimension.

```
int x[2][3][4];


for (int i=0; i<2; i++) {
    for (int j=0; j<3; j++) {
        for (int k=0; k<4; k++) {
            x[i][j][k] = (some_value);
        }
    }
}
```

Accessing elements in Three-Dimensional Array in C

Accessing elements in 3D Arrays is also similar to that of 3D Arrays. The difference is we have to use three loops instead of two loops for one additional dimension in 3D Arrays.

Syntax:

```
array_name[x][y][z]
```

where,

- **x:** Index of 2D array.

- **y:** Index of that 2D array row.
- **z:** Index of that 2D array column.

- C

```c
// C program to print elements of Three-Dimensional Array

#include <stdio.h>

int main(void)
{
    // initializing the 3-dimensional array
    int x[2][3][2] = { { { 0, 1 }, { 2, 3 }, { 4, 5 } },
                       { { 6, 7 }, { 8, 9 }, { 10, 11 } } };

    // output each element's value
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 3; ++j) {
            for (int k = 0; k < 2; ++k) {
                printf("Element at x[%i][%i][%i] = %d\n", i,
                       j, k, x[i][j][k]);
            }
        }
    }
    return (0);
}
```

Output

```
Element at x[0][0][0] = 0
Element at x[0][0][1] = 1
Element at x[0][1][0] = 2
Element at x[0][1][1] = 3
Element at x[0][2][0] = 4
Element at x[0][2][1] = 5
Element at x[1][0][0] = 6
Element at x[1][0][1] = 7
Element at x[1][1][0] = 8
Element at x[1][1][1] = 9
Element at x[1][2][0] = 10
Element at x[1][2][1] = 11
```

In similar ways, we can create arrays with any number of dimensions. However, the complexity also increases as the number of dimensions increases. The most used multidimensional array is the Two-Dimensional Array.

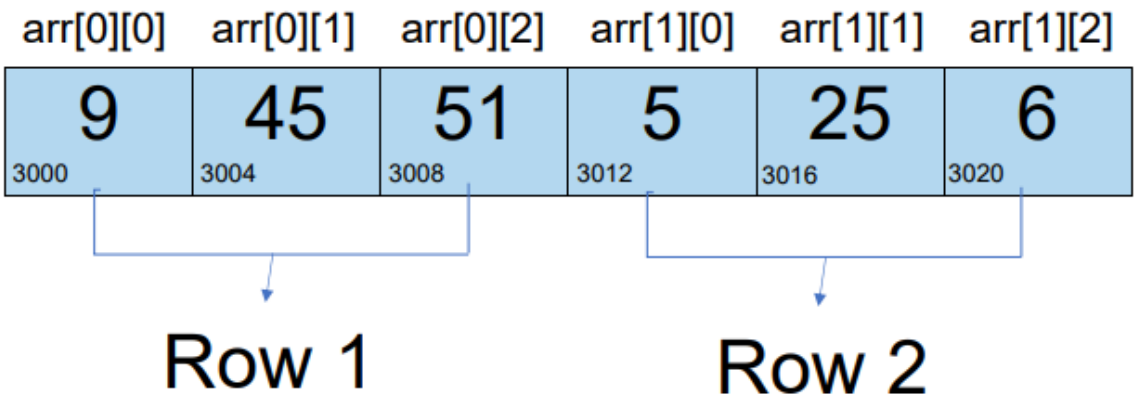
Arrays are also closely related to pointers in C language. To know more about the Relationship of Arrays with Pointers in C, refer to this article.

Last Updated : 21 Mar, 2023
473

Similar Reads

Initialization of a multidimensional arrays in C/C++

Multidimensional Pointer Arithmetic in C/C++



Difference between multidimensional array in C++ and Java

How to print dimensions of multidimensional array in C++

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]