

# Array in C programming

## What is an Array?

An array in C language is a collection of similar data-type, means an array can hold value of a particular data type for which it has been declared. Arrays can be created from any of the C data-types int, float, and char. So an integer array can only hold integer values and cannot hold values other than integer. When we declare array, it allocates contiguous memory location for storing values whereas 2 or 3 variables of same data-type can have random locations. So this is the most important difference between a variable and an array.

### Types of Arrays:

1. One dimension array (Also known as 1-D array).
2. Two dimension array (Also known as 2-D array).
3. Multi-dimension array.

**Decla Syntax:** data\_type array\_name[width];

**Example:** int roll[8];

In our example, int specifies the type of the variable, roll specifies the name of the variable and the value in bracket [8] is new for newbie. The bracket ([ ]) tells compiler that it is an array and number mention in the bracket specifies that how many elements (values in any array is called elements) it can store. This number is called dimension of array.

So, with respect to our example we have declared an array of integer type and named it "roll" which can store roll numbers of 8 students. You can see memory arrangement of above declared array in the following image:

Name	roll[0]	roll[1]	roll[2]	roll[3]	roll[4]	roll[5]	roll[6]	roll[7]
Values	12	45	32	23	17	49	5	11
Address	1000	1002	1004	1006	1008	1010	1012	1014

1-D Array memory arrangement

## C Array Assignment and Initialization:

We can initialize and assign values to the arrays in the same way as we do with variable. We can assign value to an array at the time of declaration or during runtime. Let's look at each approach.

**Syntax:** data\_type array\_name[size]={list of values};

**Example:**

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

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

In our above array example we have declared an integer array and named it "arr" which can hold 5 elements, we are also initializing arrays in the same time.

Both statements in our example are valid method to declare and initialize single dimension array. In our first example we mention the size (5) of an array and assigned it values in curly brace, separating element's value by comma (,). But in second example we left the size field blank but we provided its element's value. When we only give element values without providing size of an array then C compiler automatically assumes its size from given element values.

There is one more method to initialize array C programming; in this method we can assign values to individual element of an array. For this let's look at example:

## Array Initialization Example

[view plain](#)

```
1. #include<stdio.h>
2. #include<conio.h>
3.
4. void main()
5. {
6.     int arr[5],i;
7.     clrscr();
8.     arr[0]=10;
9.     arr[1]=20;
10.    arr[2]=30;
11.    arr[3]=40;
12.    arr[4]=50;
13.
14.    printf("Value in array arr[0] : %d\n",arr[0]);
15.    printf("Value in array arr[1] : %d\n",arr[1]);
16.    printf("Value in array arr[2] : %d\n",arr[2]);
17.    printf("Value in array arr[3] : %d\n",arr[3]);
18.    printf("Value in array arr[4] : %d\n",arr[4]);
19.    printf("\n");
20.
21.    for(i=0;i<5;i++)
22.    {
23.        printf("Value in array arr[%d] : %d\n",i,arr[i]);
24.    }
25.    getch();
26. }
```

In the above c arrays example we have assigned the value of integer array individually like we do with an integer variable. We have called array element's value individually and using for loop so that it would be clear for beginner and semi-beginner C programmers. So, from the above example it is evident that we can assign values to an array element individually and can call them individually whenever we need them.

## How to work with Two Dimensional Arrays in C

### Declaration of 2D array:

**Syntax:** data\_type array\_name[row\_size][column\_size];

**Example:** int arr[3][3];

So the above example declares a 2D array of integer type. This integer array has been named arr and it can hold up to 9 elements (3 rows x 3 columns).

### 2D Array

arr	col[0]	col[1]	col[2]
row[0]	10	20	45
row[1]	42	79	81
row[2]	89	9	36

**2D Array Arrangement**

### Memory Map of 2D Array

arr[0][0]	arr[0][1]	arr[0][2]	arr[1][0]	arr[1][1]	arr[1][2]	arr[2][0]	arr[2][1]	arr[2][2]
12	45	63	89	34	73	19	76	49
1000	1002	1004	1006	1008	1010	1012	1014	1016

**Memory Map of 2 Dimensional Array**

### Code for assigning & displaying 2D Array

[view plain](#)

```
1. #include<stdio.h>
2. #include<conio.h>
3.
4. void main()
5. {
6.     int i, j;
7.     int arr[3][3]={
8.         {12, 45, 63},
9.         {89, 34, 73},
10.        {19, 76, 49}
11.    };
12. clrscr();
13. printf(":::2D Array Elements:::\n\n");
14. for(i=0;i<3;i++)
15. {
16.     for(j=0;j<3;j++)
17.     {
18.         printf("%d\t",arr[i][j]);
19.     }
20.     printf("\n");
21. }
22. getch();
23. }
```

So in the above example we have declared a 2D array named arr which can hold 3x3 elements. We have also initialized that array with values, because we told the compiler that this array will contain 3 rows (0 to 2) so we divided elements accordingly. Elements for column have been differentiated by a comma (.). When compiler finds comma in array elements then it assumes comma as beginning of next element value. We can also define the same array in other ways, like.  
int arr[3][3]={12, 45, 63, 89, 34, 73, 19, 76, 49}; or,  
int arr[ ][3]={12, 45, 63, 89, 34, 73, 19, 76, 49};

But this kind of declaration is not acceptable in C language programming.

int arr[2][ ]={12, 45, 63, 89, 34, 73, 19, 76, 49}; or,  
int arr[ ][ ]={12, 45, 63, 89, 34, 73, 19, 76, 49};

To display 2D array elements we have to just point out which element value we want to display. In our example we have a arr[3][3], so the array element reference will be from arr[0][0] to arr[2][2]. We can print display any element from this range. But in our example I have used for loop for my convenience, otherwise I had to write 9 printf statements to display all elements of array. So for loop i handles row of 2D array and for loop j handles column. I have formatted the output display of array so that we can see the elements in tabular form.

## How to work with Multidimensional Array in C Programming

C allows array of two or more dimensions and maximum numbers of dimension a C program can have is depend on the compiler we are using. Generally, an array having one dimension is called 1D array, array having two dimensions called 2D array and so on. So in C programming an array can have two or three or four or even ten or more dimensions. More dimensions in an array means more data it can hold and of course more difficulties to manage and understand these arrays. A multidimensional array has following syntax:

### Syntax:

**type array\_name[d1][d2][d3][d4].....[dn];**

Where dn is the size of last dimension.

### Example:

**int table[5][5][20];**

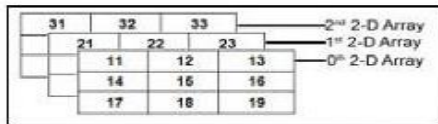
**float arr[5][6][5][6][5];**

In our example array "table" is a 3D (A 3D array is an array of arrays of arrays.) array which can hold 500 integer type elements. And array "arr" is a 5D array which can hold 4500 floating-point elements. Can see the power of array over variable? When it comes to hold multiple values in a C programming, we need to declare several variables (for example to store 150 integers) but in case of array, a single array can hold thousands of values (depending on compiler, array type etc).

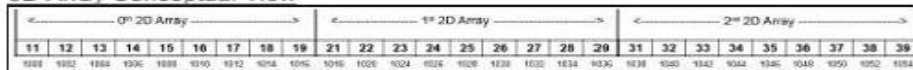
**Note: To make this multidimensional array example simple I will discuss 3D array for the sake of simplicity. Once you grab the logic how 3D array works then you can handle 4D array or any multidimensional array easily.**

## How to Declaration and Initialization 3D Array

Before we move to serious programming let's have a look of 3D array. A 3D array can be assumed as an array of arrays of arrays, it is array (collection) of 2D arrays and as you know 2D array itself is array of 1D array. It sounds a bit confusing but don't worry as you will lead your learning on multidimensional array, you will grasp all logic and concept. A diagram can help you to understand this.



3D Array Conceptual View



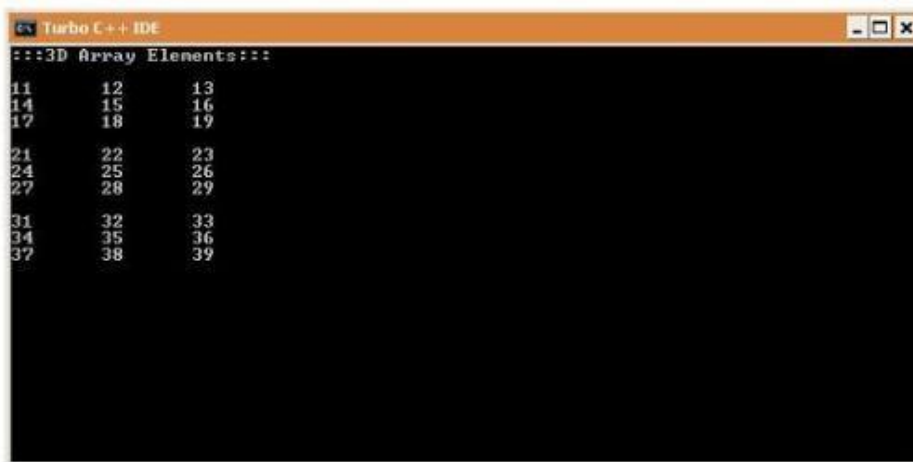
3D array memory map.

We can initialize a 3D array at the compile time as we initialize any other variable or array, by default an uninitialized 3D array contains garbage value. Let's see a complete example on how we can work with a 3D array.

### Example of Declaration and Initialization 3D Array

```
#include<stdio.h>
1. #include<conio.h>
2.
3. void main()
4. {
5.     int i, j, k;
6.     int arr[3][3][3]=
7.     {
8.         {
9.             {11, 12, 13},
10.            {14, 15, 16},
11.            {17, 18, 19}
12.        },
13.        {
14.            {21, 22, 23},
15.            {24, 25, 26},
16.            {27, 28, 29}
17.        },
18.        {
19.            {31, 32, 33},
20.            {34, 35, 36},
21.            {37, 38, 39}
22.        },
23.    };
24. clrscr();
25. printf(":::3D Array Elements:::\n\n");
26. for(i=0;i<3;i++)
27. {
28.     for(j=0;j<3;j++)
29.     {
30.         for(k=0;k<3;k++)
31.         {
32.             printf("%d\t",arr[i][j][k]);
33.         }
34.         printf("\n");
35.     }
36.     printf("\n");
37. }
38. getch();
39. }
```



A screenshot of the Turbo C++ IDE window. The title bar says "Turbo C++ IDE". The main window displays the text "3D Array Elements:" followed by a 3x3 grid of numbers. The numbers are arranged in three rows and three columns. The first row contains 11, 12, 13. The second row contains 14, 15, 16. The third row contains 17, 18, 19. The fourth row contains 21, 22, 23. The fifth row contains 24, 25, 26. The sixth row contains 27, 28, 29. The seventh row contains 31, 32, 33. The eighth row contains 34, 35, 36. The ninth row contains 37, 38, 39.

```
3D Array Elements:
11    12    13
14    15    16
17    18    19

21    22    23
24    25    26
27    28    29

31    32    33
34    35    36
37    38    39
```

So in the above example we have declared multidimensional array and named this integer array as "arr" which can hold 3x3x3 (27 integers) elements. We have also initialized multidimensional array with some integer values.

As I told you earlier that a 3D array is array of 2D array therefore I have divided elements accordingly so that you can get 3D array better and understand it easily. See the C code sample above, line no. 9-13, 14-18 and 19-23, each block is a 2D array and collectively from line no. 2-24 makes a 3D array. You can also assign values to this multidimensional array in other way like this.

```
int arr[3][3][3] = {11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39};
```

This kind of C multidimensional array (3D array) declaration is quite confusing for new C programmers; you cannot guess location of array element by just looking at the declaration. But look at the above multidimensional array example where you can get a clear idea about each element location. For example, consider 3D array as a collection of tables, to access or store any element in a 3D array you need to know first table number then row number and lastly column number. For instance you need to access value 25 from above 3D array. So, first check the table (among 3 tables which table has the value), once you find the table number now check which row of that table has the value again if you get the row no then check column number and you will get the value. So applying above logic, 25 located in table no. 1 row no. 1 and column no. 1, hence the address is arr[1][1][1]. Print this address and you will get the output.

So the conceptual syntax for 3D array stands like this.

```
data_type array_name[table][row][column];
```

If you want to store values in any 3D array then first point to table number, row number and lastly to column number.

```
arr[0][1][2] = 32;
arr[1][0][1] = 49;
```

We know [how to work with an array \(1D array\) having one dimension](#). In C language it is possible to have more than one dimension in an array. In this tutorial we are going to learn how we can use two dimensional arrays (2D arrays) to store values. Because it is a 2D array so its structure will be different from one dimension array. The 2D array is also known as Matrix or Table, it is an array of array. See the below image, here each row is an array.

Above code is for assigning values at particular location of an array but if you want to store value in continuous location of array then you should use loop. Here is an example using for loop.

```
1. #include<stdio.h>
2. #include<conio.h>
3.
4. void main()
5. {
6.     int i, j, k, x=1;
7.     int arr[3][3][3];
8.     clrscr();
9.     printf(":::3D Array Elements:::\n\n");
10.
11.     for(i=0;i<3;i++)
12.     {
13.         for(j=0;j<3;j++)
14.         {
15.             for(k=0;k<3;k++)
16.             {
17.                 arr[i][j][k] = x;
18.                 printf("%d\t",arr[i][j][k]);
19.                 x++;
20.             }
21.             printf("\n");
22.         }
23.         printf("\n");
24.     }
25.     getch();
26. }
```