



- Dimensional variables : arrays and matrices
- One Dimensional array
- two Dimensional array, square array(as special state of two Dimensional array)

الثاني عشر –  
الثالث عشر

A one-dimensional array is a group of elements having the same datatype and same name. Individual elements are referred to using common name and unique index of the elements.

The simplest form of an array is one-dimensional-array. The array itself is given name and its elements are referred to by their subscripts. In C++, an array is denoted as follows:

```
array_name[array_size]
```

where size specifies the number of elements in the array and the subscript (also called index) value ranges from 0 through size-1.

## Initializing Arrays

You can initialize C++ array elements either one by one or using a single statement as follows –

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array –

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –

```
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

You will create exactly the same array as you did in the previous example.

```
balance[4] = 50.0;
```

The above statement assigns element number 5<sup>th</sup> in the array a value of 50.0. Array with 4<sup>th</sup> index will be 5<sup>th</sup>, i.e., last element because all arrays have 0 as the index of their first



element which is also called base index. Following is the pictorial representation of the same array we discussed above –

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

## Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –

```
double salary = balance[9];
```

The above statement will take 10<sup>th</sup> element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays –

[Live Demo](#)

```
#include <iostream>
using namespace std;

#include <iomanip>
using std::setw;

int main () {

    int n[ 10 ]; // n is an array of 10 integers

    // initialize elements of array n to 0
    for ( int i = 0; i < 10; i++ ) {
        n[ i ] = i + 100; // set element at location i to i + 100
    }
    cout << "Element" << setw( 13 ) << "Value" << endl;

    // output each array element's value
    for ( int j = 0; j < 10; j++ ) {
        cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;
    }

    return 0;
}
```

This program makes use of **setw()** function to format the output. When the above code is compiled and executed, it produces the following result –



Element	Value
0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108
9	109

## C++ Multi-dimensional Arrays

---

C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration –

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

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array –

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

### Two-Dimensional Arrays

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows –

```
type arrayName [ x ][ y ];
```

Where **type** can be any valid C++ data type and **arrayName** will be a valid C++ identifier.

A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array **a**, which contains three rows and four columns can be shown as below –



	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Thus, every element in array `a` is identified by an element name of the form `a[i][j]`, where `a` is the name of the array, and `i` and `j` are the subscripts that uniquely identify each element in `a`.

## Initializing Two-Dimensional Arrays

Multidimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 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 nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example –

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

## Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example –

```
int val = a[2][3];
```

The above statement will take 4<sup>th</sup> element from the 3<sup>rd</sup> row of the array. You can verify it in the above diagram.



```
#include <iostream>
using namespace std;

int main () {
    // an array with 5 rows and 2 columns.
    int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};

    // output each array element's value
    for ( int i = 0; i < 5; i++ )
        for ( int j = 0; j < 2; j++ ) {

            cout << "a[" << i << "][" << j << "]: ";
            cout << a[i][j]<< endl;
        }

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.