# Array

## Definition

- Arrays are defined as the collection of similar type of data items stored at contiguous memory locations.
- Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.
- Array is the simplest data structure where each data element can be randomly accessed by using its index number.
- For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variable for the marks in different subject. instead of that, we can define an array which can store the marks in each subject at a the contiguous memory locations.

The array **marks[10]** defines the marks of the student in 10 different subjects where each subject marks are located at a particular subscript in the array i.e. **marks[0]** denotes the marks in first subject, **marks[1]** denotes the marks in 2nd subject and so on.

## Properties of the Array

1. Each element is of same data type and carries a same size i.e. int = 4 bytes.
2. Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
3. Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of data element.

for example, in C language, the syntax of declaring an array is like following:

```
1. int arr[10]; char arr[10]; float arr[5]
```

## Need of using Array

In computer programming, the most of the cases requires to store the large number of data of similar type. To store such amount of data, we need to define a large number of variables. It would be very difficult to remember names of all the variables while writing the programs. Instead of naming all the variables with a different name, it is better to define an array and store all the elements into it.

Following example illustrates, how array can be useful in writing code for a particular problem.

In the following example, we have marks of a student in six different subjects. The problem intends to calculate the average of all the marks of the student.

In order to illustrate the importance of array, we have created two programs, one is without using array and other involves the use of array to store marks.

**Program without array:**

```
1. #include <stdio.h>
2. void main ()
3. {
4.     int marks_1 = 56, marks_2 = 78, marks_3 = 88, marks_4 = 76, marks_5 = 56, marks_6 = 89;
5.     float avg = (marks_1 + marks_2 + marks_3 + marks_4 + marks_5 +marks_6) / 6 ;
6.     printf(avg);
7. }
```

**Program by using array:**

```
1. #include <stdio.h>
2. void main ()
3. {
4.     int marks[6] = {56,78,88,76,56,89);
5.     int i;
```

```
6.      float avg;
7.      for (i=0; i<6; i++ )
8.      {
9.          avg = avg + marks[i];
10.     }
11.     printf(avg);
12. }
```

## Complexity of Array operations

Time and space complexity of various array operations are described in the following table.

### Time Complexity

| Algorithm | Average Case | Worst Case |
|---|---|---|
| Access | O(1) | O(1) |
| Search | O(n) | O(n) |
| Insertion | O(n) | O(n) |
| Deletion | O(n) | O(n) |

#### Space Complexity

In array, space complexity for worst case is **O(n)**.

## Advantages of Array

- Array provides the single name for the group of variables of the same type therefore, it is easy to remember the name of all the elements of an array.
- Traversing an array is a very simple process, we just need to increment the base address of the array in order to visit each element one by one.
- Any element in the array can be directly accessed by using the index.

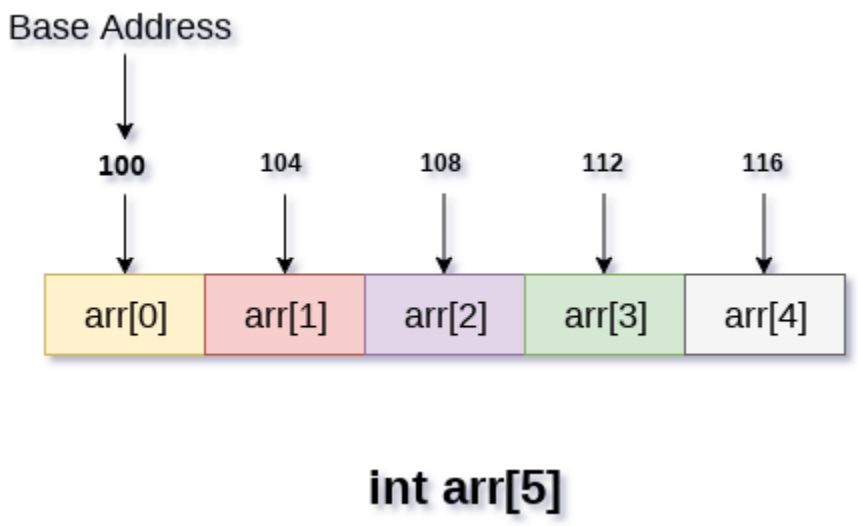
## Memory Allocation of the array

As we have mentioned, all the data elements of an array are stored at contiguous locations in the main memory. The name of the array represents the base address or the address of first element in the main memory. Each element of the array is represented by a proper indexing.

The indexing of the array can be defined in three ways.

1. 0 (zero - based indexing) : The first element of the array will be arr[0].
2. 1 (one - based indexing) : The first element of the array will be arr[1].
3. n (n - based indexing) : The first element of the array can reside at any random index number.

In the following image, we have shown the memory allocation of an array arr of size 5. The array follows 0-based indexing approach. The base address of the array is 100th byte. This will be the address of arr[0]. Here, the size of int is 4 bytes therefore each element will take 4 bytes in the memory.




In 0 based indexing, If the size of an array is n then the maximum index number, an element can have is **n-1**. However, it will be n if we use **1** based indexing.

## Accessing Elements of an array

To access any random element of an array we need the following information:

1. Base Address of the array.
2. Size of an element in bytes.
3. Which type of indexing, array follows.

Address of any element of a 1D array can be calculated by using the following formula:

1. Byte address of element A[i]  = base address + size * ( i - first index)

**Example :**

1. In an array, A[-10 ..... +2 ], Base address (BA) = 999, size of an element = 2 bytes,
2. find the location of A[-1].
3. L(A[-1]) = 999 + [(-1) - (-10)] x 2
4. = 999 + 18
5. = 1017

## Passing array to the function :

As we have mentioned earlier that, the name of the array represents the starting address or the address of the first element of the array. All the elements of the array can be traversed by using the base address.

The following example illustrate, how the array can be passed to a function.

**Example:**

```
1. #include <stdio.h>
2. int summation(int[]);
3. void main ()
4. {
5.     int arr[5] = {0,1,2,3,4};
6.     int sum = summation(arr);
7.     printf("%d",sum);
8. }
9.
10. int summation (int arr[])
11. {
12.     int sum=0,i;
13.     for (i = 0; i<5; i++)
14.     {
15.         sum = sum + arr[i];
16.     }
```

```
17.     return sum;
18. }
```

The above program defines a function named as summation which accepts an array as an argument. The function calculates the sum of all the elements of the array and returns it.

# 2D Array

2D array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

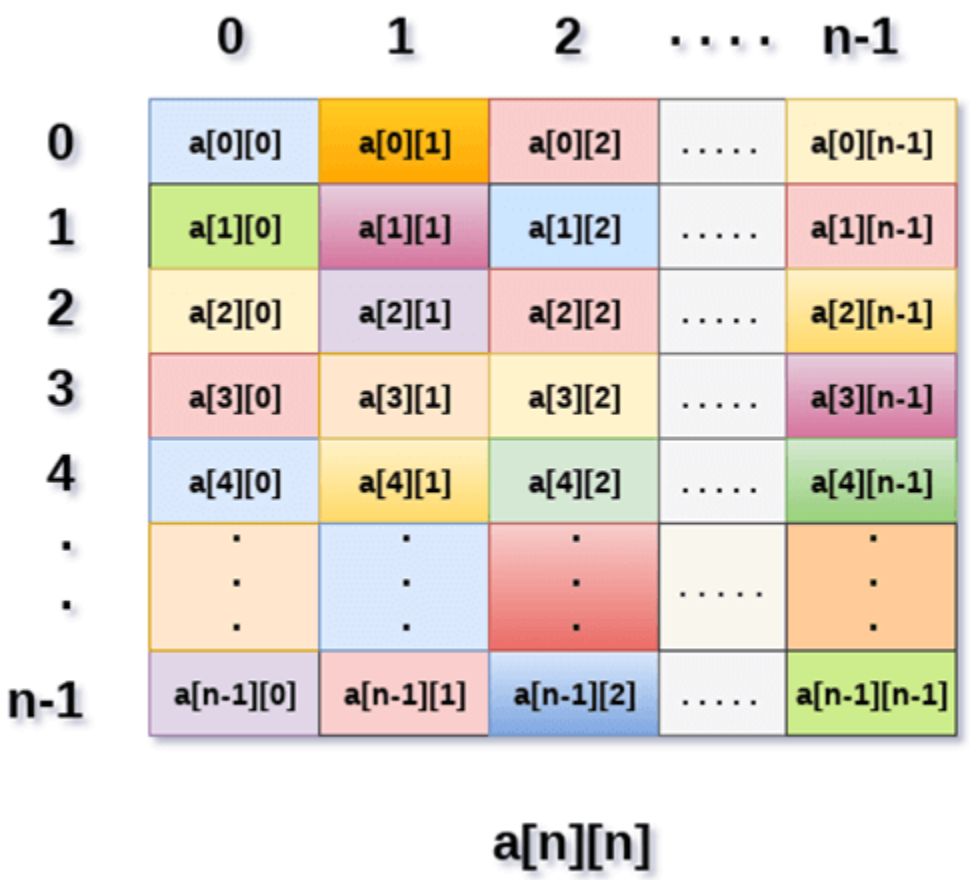
However, 2D arrays are created to implement a relational database look alike data structure. It provides ease of holding bulk of data at once which can be passed to any number of functions wherever required.

## How to declare 2D Array

The syntax of declaring two dimensional array is very much similar to that of a one dimensional array, given as follows.

```
1.  int arr[max_rows][max_columns];
```

however, It produces the data structure which looks like following.




Above image shows the two dimensional array, the elements are organized in the form of rows and columns. First element of the first row is represented by a[0][0] where the number shown in the first index is the number of that row while the number shown in the second index is the number of the column.

## How do we access data in a 2D array

Due to the fact that the elements of 2D arrays can be random accessed. Similar to one dimensional arrays, we can access the individual cells in a 2D array by using the indices of the cells. There are two indices attached to a particular cell, one is its row number while the other is its column number.

However, we can store the value stored in any particular cell of a 2D array to some variable x by using the following syntax.

```
1.  int x = a[i][j];
```

where i and j is the row and column number of the cell respectively.

We can assign each cell of a 2D array to 0 by using the following code:

```
1.  for ( int i=0; i<n ;i++)
2.  {
3.      for (int j=0; j<n; j++)
```

```
4.      {
5.          a[i][j] = 0;
6.      }
7.  }
```

## Initializing 2D Arrays

We know that, when we declare and initialize one dimensional array in C programming simultaneously, we don't need to specify the size of the array. However this will not work with 2D arrays. We will have to define at least the second dimension of the array.

The syntax to declare and initialize the 2D array is given as follows.

```
1.  int arr[2][2] = {0,1,2,3};
```

The number of elements that can be present in a 2D array will always be equal to (**number of rows * number of columns**).

**Example :** Storing User's data into a 2D array and printing it.

**C Example :**

```
1.  #include <stdio.h>
2.  void main ()
3.  {
4.      int arr[3][3],i,j;
5.      for (i=0;i<3;i++)
6.      {
7.          for (j=0;j<3;j++)
8.          {
9.              printf("Enter a[%d][%d]: ",i,j);
10.             scanf("%d",&arr[i][j]);
11.         }
12.     }
13.     printf("\n printing the elements ....\n");
14.     for(i=0;i<3;i++)
15.     {
16.         printf("\n");
17.         for (j=0;j<3;j++)
18.         {
19.             printf("%d\t",arr[i][j]);
20.         }
21.     }
22. }
```

### Java Example

```
1.  import java.util.Scanner;
2.  publicclass TwoDArray {
3.  publicstaticvoid main(String[] args) {
4.      int[][] arr = newint[3][3];
5.      Scanner sc = new Scanner(System.in);
6.      for (inti =0;i<3;i++)
7.      {
8.          for(intj=0;j<3;j++)
9.          {
10.             System.out.print("Enter Element");
11.              arr[i][j]=sc.nextInt();
12.             System.out.println();
13.         }
14.     }
15.     System.out.println("Printing Elements...");
```

```
16.     for(inti=0;i<3;i++)
17.     {
18.         System.out.println();
19.          for(intj=0;j<3;j++)
20.         {
21.             System.out.print(arr[i][j]+"\t");
22.         }
23.     }
24. }
25. }
```

## C# Example

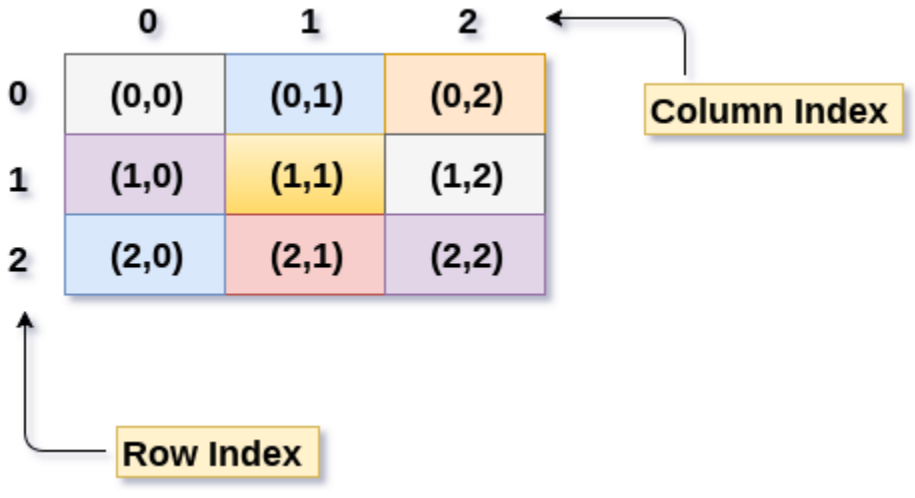
```
1.  using System;
2.
3.  public class Program
4.  {
5.      public static void Main()
6.      {
7.          int[,] arr = new int[3,3];
8.          for (int i=0;i<3;i++)
9.          {
10.             for (int j=0;j<3;j++)
11.             {
12.                 Console.WriteLine("Enter Element");
13.                  arr[i,j]= Convert.ToInt32(Console.ReadLine());
14.             }
15.         }
16.         Console.WriteLine("Printing Elements...");
17.         for (int i=0;i<3;i++)
18.         {
19.             Console.WriteLine();
20.             for (int j=0;j<3;j++)
21.             {
22.                 Console.Write(arr[i,j]+" ");
23.             }
24.         }
25.     }
26. }
```

# Mapping 2D array to 1D array

When it comes to map a 2 dimensional array, most of us might think that why this mapping is required. However, 2 D arrays exists from the user point of view. 2D arrays are created to implement a relational database table lookalike data structure, in computer memory, the storage technique for 2D array is similar to that of an one dimensional array.

The size of a two dimensional array is equal to the multiplication of number of rows and the number of columns present in the array. We do need to map two dimensional array to the one dimensional array in order to store them in the memory.

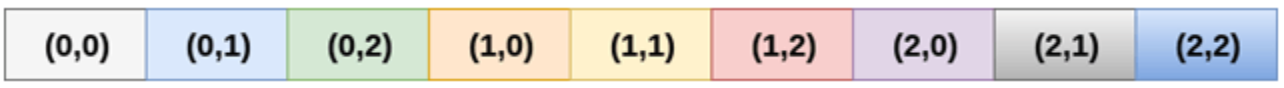
A 3 X 3 two dimensional array is shown in the following image. However, this array needs to be mapped to a one dimensional array in order to store it into the memory.



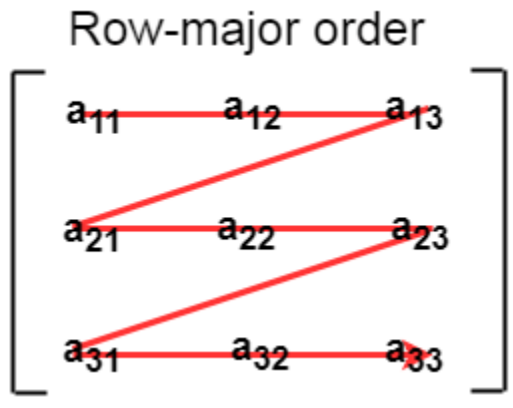

There are two main techniques of storing 2D array elements into memory

## 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.

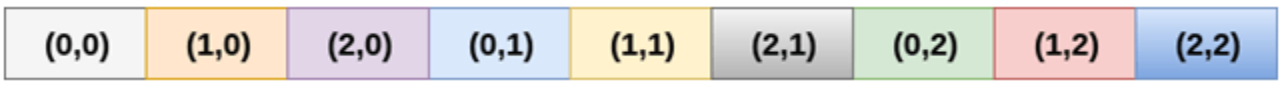



first, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.

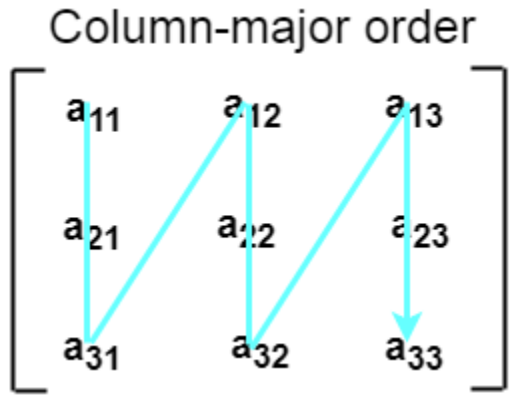



## 2. Column Major ordering

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.




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,

1. 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 :**

1. a[10...30, 55...75], base address of the array (BA) = 0, size of an element = 4 bytes .
2. Find the location of a[15][68].
3. 
4. Address(a[15][68]) = 0 +
5. ((15 - 10) x (68 - 55 + 1) + (68 - 55)) x 4
6. 
7. = (5 x 14 + 13) x 4
8. = 83 x 4
9. = 332 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,

1. Address(a[i][j]) = ((j*m)+i)*Size + BA

where BA is the base address of the array.

**Example:**

1. A [-5 ... +20][20 ... 70], BA = 1020, Size of element = 8 bytes. Find the location of a[0][30].
2. 
3. Address [A[0][30]) = ((30-20) x 24 + 5)  x 8 + 1020  =  245 x 8 + 1020 = 2980 bytes