ARRAY

An array is a group (or collection) of same data types. For example an int array holds the elements of int types while a float array holds the elements of float types.

Why we need Array in C Programming?

Consider a scenario where you need to find out the average of 100 integer numbers entered by user. In C, you have two ways to do this: 1) Define 100 variables with int data type and then perform 100 scanf() operations to store the entered values in the variables and then at last calculate the average of them. 2) Have a single integer array to store all the values, loop the array to store all the entered values in array and later calculate the average.

Which solution is better according to you? Obviously the second solution, it is convenient to store same data types in one single variable and later access them using array index (we will discuss that later).

How to declare Array in C

```
int num[35];  /* An integer array of 35 elements */
char ch[10];  /* An array of characters for 10 elements */
```

Similarly an array can be of any data type such as double, float, short etc.

How to access element of an array in C

You can use **array subscript** (or index) to access any element stored in array. Subscript starts with 0, which means arr[0] represents the first element in the array arr.

In general arr[n-1] can be used to access nth element of an array. where n is any integer number.

For example:

```
int mydata[20];
mydata[0] /* first element of array mydata*/
mydata[19] /* last (20th) element of array mydata*/
```

Example of Array In C programming to find out the average of 4 integers

```
#include <stdio.h>
int main()
{
   int avg = 0;
   int sum =0;
   int x=0;

   /* Array- declaration – length 4*/
   int num[4];

   /* We are using a for loop to traverse through the array
    * while storing the entered values in the array
    */
   for (x=0; x<4;x++)
   {
      printf("Enter number %d \n", (x+1));
      scanf("%d", &num[x]);
   }
   for (x=0; x<4;x++)
```

```
   {
      sum = sum+num[x];
   }

   avg = sum/4;
   printf("Average of entered number is: %d", avg);
   return 0;
}
```

Output:

```
Enter number 1
10
Enter number 2
10
Enter number 3
20
Enter number 4
40
Average of entered number is: 20
```

Lets discuss the important parts of the above program:

Input data into the array

Here we are **iterating the array** from 0 to 3 because the size of the array is 4. Inside the loop we are displaying a message to the user to enter the values. All the input values are stored in the corresponding array elements using scanf function.

```
for (x=0; x<4;x++)
{
   printf("Enter number %d \n", (x+1));
   scanf("%d", &num[x]);
}
```

Reading out data from an array

Suppose, if we want to display the elements of the array then we can use the for loop in C like this.

```
for (x=0; x<4;x++)
{
   printf("num[%d]\n", num[x]);
}
```

Various ways to initialize an array

In the above example, we have just declared the array and later we initialized it with the values input by user. However you can also initialize the array during declaration like this:

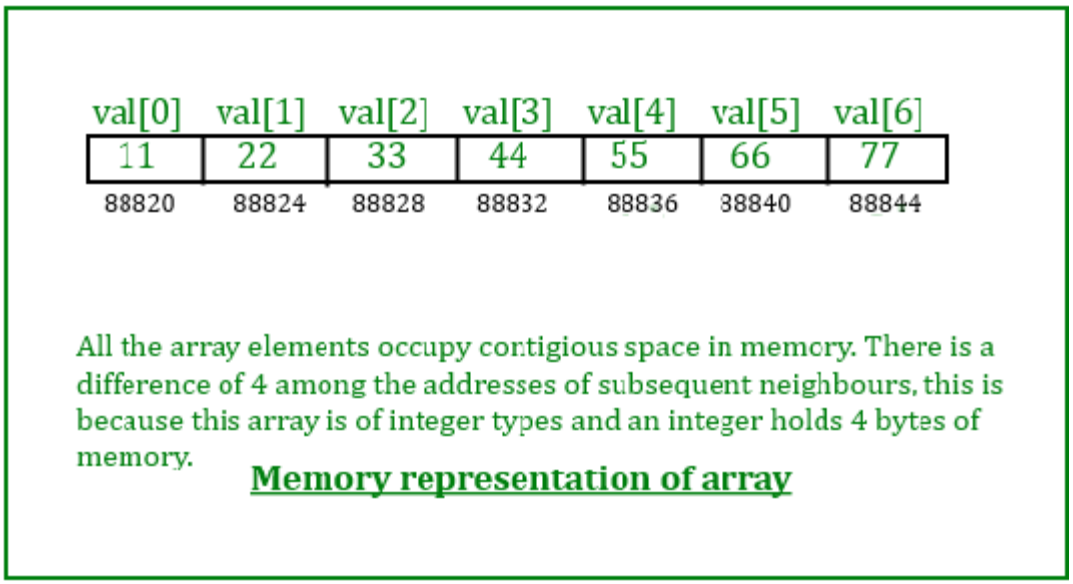
```
int arr[5] = {1, 2, 3, 4 ,5};
```

OR (both are same)

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

Un-initialized array always contain garbage values.

C Array – Memory representation



More Topics on Arrays in C:
2D array – We can have multidimensional arrays in C like 2D and 3D array. However the most popular and frequently used array is 2D – two dimensional array. In this post you will learn how to declare, read and write data in 2D array along with various other features of it.

Passing an array to a function– Generally we pass values and variables while calling a function, likewise we can also pass arrays to a function. You can pass array's element as well as whole array (by just specifying the array name, which works as a pointer) to a function.

Pointer to array – Array elements can be accessed and manipulated using pointers in C. Using pointers you can easily handle array. You can have access of all the elements of an array just by assigning the array's base address to pointer variable.

Two dimensional (2D) arrays in C programming with example:

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. Before we discuss more about two Dimensional array lets have a look at the following C program.

Simple Two dimensional(2D) Array Example

For now don't worry how to initialize a two dimensional array, we will discuss that part later. This program demonstrates how to store the elements entered by user in a 2d array and how to display the elements of a two dimensional array.

```
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int disp[2][3];
```

```
  /*Counter variables for the loop*/
  int i, j;
  for(i=0; i<2; i++) {
    for(j=0;j<3;j++) {
      printf("Enter value for disp[%d][%d]:", i, j);
      scanf("%d", &disp[i][j]);
    }
  }
  //Displaying array elements
  printf("Two Dimensional array elements:\n");
  for(i=0; i<2; i++) {
    for(j=0;j<3;j++) {
      printf("%d ", disp[i][j]);
      if(j==2){
        printf("\n");
      }
    }
  }
  return 0;
}
```

Output:

```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

Initialization of 2D Array

There are two ways to initialize a two Dimensional arrays during declaration.

```
int disp[2][4] = {
   {10, 11, 12, 13},
   {14, 15, 16, 17}
};
```

OR

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Although both the above declarations are valid, I recommend you to use the first method as it is more readable, because you can visualize the rows and columns of 2d array in this method.

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 }
```

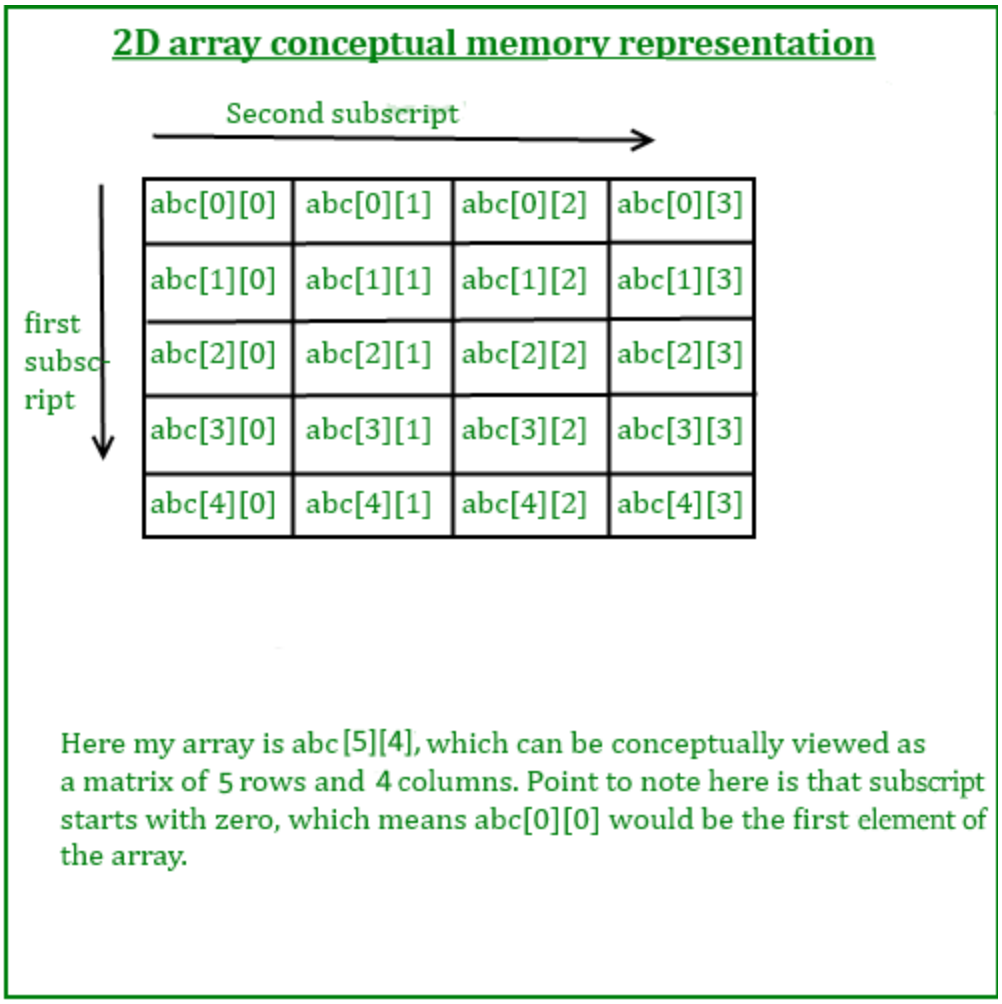
How to store user input data into 2D array

We can calculate how many elements a two dimensional array can have by using this formula:
The array arr[n1][n2] can have n1*n2 elements. The array that we have in the example below is having the dimensions 5 and 4. These dimensions are known as subscripts. So this array has **first subscript** value as 5 and **second subscript** value as 4.
So the array abc[5][4] can have 5*4 = 20 elements.

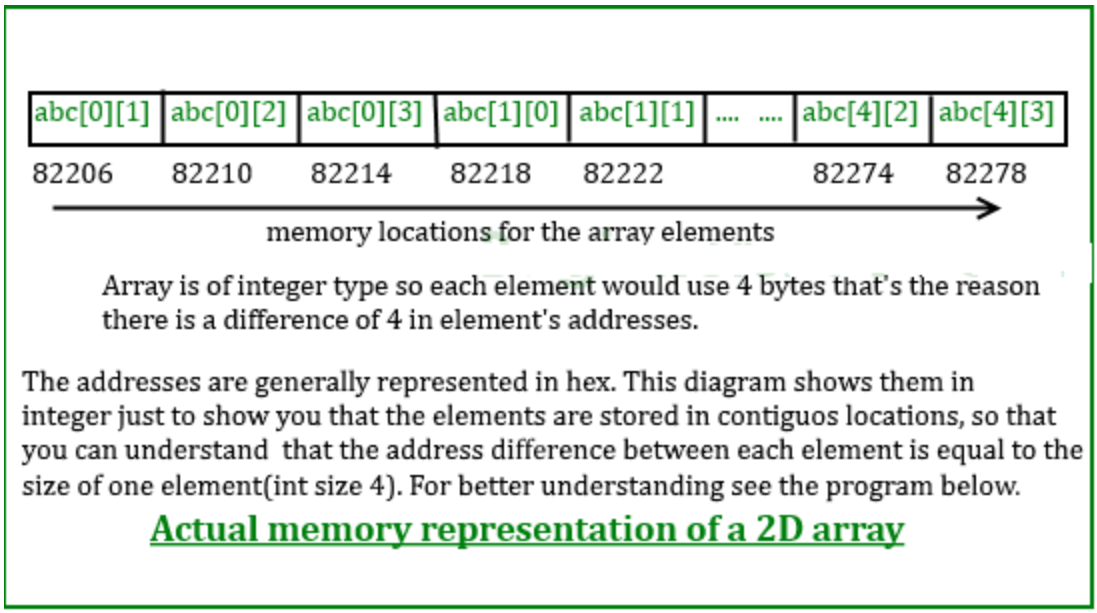
To store the elements entered by user we are using two for loops, one of them is a nested loop. The outer loop runs from 0 to the (first subscript -1) and the inner for loops runs from 0 to the (second subscript -1). This way the the order in which user enters the elements would be abc[0][0], abc[0][1], abc[0][2]…so on.

```
#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;
}
```

In above example, I have a 2D array abc of integer type. Conceptually you can visualize the above array like this:



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



Pointers & 2D array

As we know that the one dimensional array name works as a pointer to the base element (first element) of the array. However in the case 2D arrays the logic is slightly different. You can consider a 2D array as collection of several one dimensional arrays.

So abc[0] would have the address of first element of the first row (if we consider the above diagram number 1). similarly abc[1] would have the address of the first element of the second row. To understand it better, lets write a C program –

```
#include <stdio.h>
int main()
{
  int abc[5][4] ={
         {0,1,2,3},
         {4,5,6,7},
         {8,9,10,11},
         {12,13,14,15},
         {16,17,18,19}
         };
  for (int i=0; i<=4; i++)
  {
     /* The correct way of displaying an address would be
      * printf("%p ",abc[i]); but for the demonstration
      * purpose I am displaying the address in int so that
      * you can relate the output with the diagram above that
      * shows how many bytes an int element uses and how they
      * are stored in contiguous memory locations.
      *
      */
                printf("%d ",abc[i]);
  }
  return 0;
}
```

Output:

```
1600101376 1600101392 1600101408 1600101424 1600101440
```

The actual address representation should be in hex for which we use %p instead of %d, as mentioned in the comments. This is just to show that the elements are stored in contiguous memory locations. You can relate the output with the diagram above to see that the difference between these addresses is actually number of bytes consumed by the elements of that row.

The addresses shown in the output belongs to the first element of each row `abc[0][0]`, `abc[1][0]`, `abc[2][0]`, `abc[3][0]` and `abc[4][0]`.