UNIT III – ARRAYS AND STRINGS IN C

Arrays: Initialization - One dimensional, Two dimensional, and Multi-dimensional arrays - String: Basics, declaring and initializing strings, string handling functions: standard and user defined functions

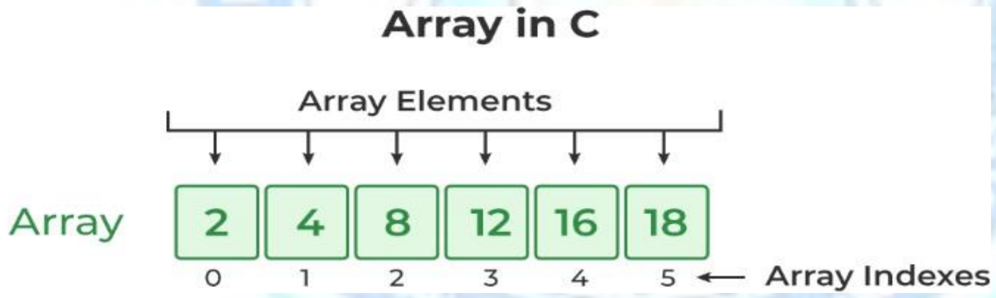
3.1 INTRODUCTION TO ARRAYS

Arrays

Array in C is one of the most used data structures in C programming. It is a simple and fast way of storing multiple values under a single name.

What is Array in C?

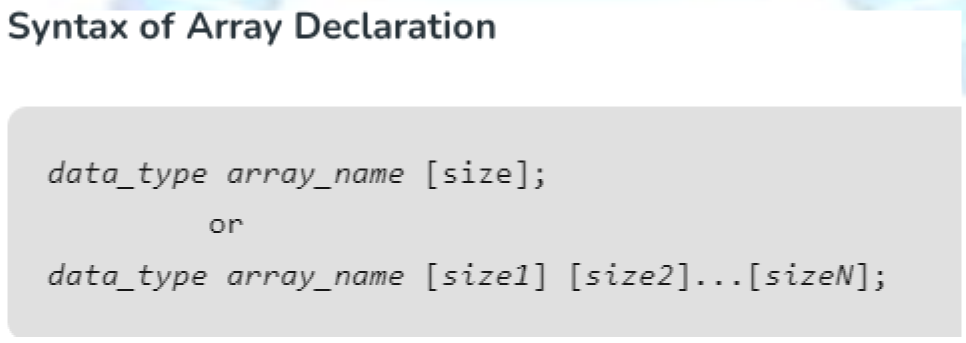
An array in C is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.



a) Array Declaration

In C, we have to declare the array like any other variable before using it. We can declare an array by specifying its name, the type of its elements, and the size of its dimensions.

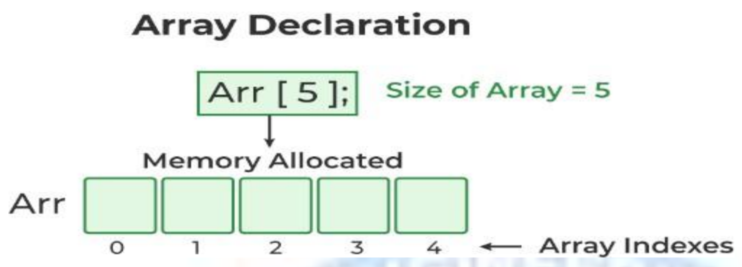
Syntax of Array Declaration



Syntax of Array Declaration

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

where N is the number of dimensions.



The C arrays are static in nature, i.e., they are allocated memory at the compile time.

Example of Array Declaration

```
// C Program to illustrate the array declaration
#include <stdio.h>
int main()
{
        // declaring array of integers
  int arr_int[5];
        // declaring array of characters
  char arr_char[5];
  return 0;
}
```

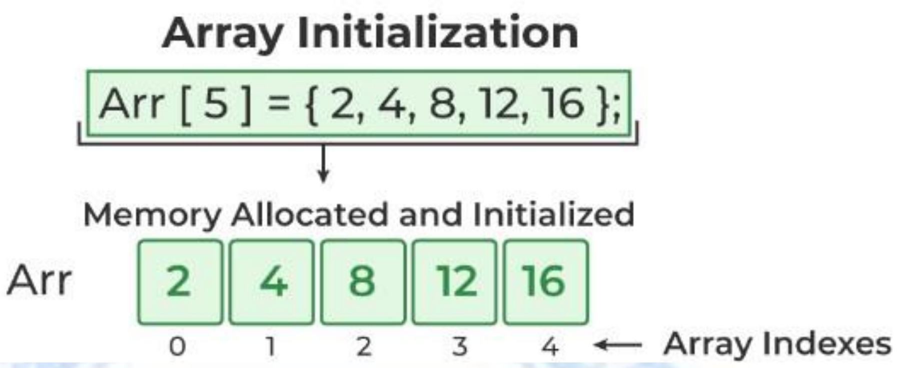
b) Array Initialization

Initialization in C is the process to assign some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value. There are multiple ways in which we can initialize an array in C.

1. Array Initialization with Declaration

In this method, we initialize the array along with its declaration. We use an initializer list to initialize multiple elements of the array. An initializer list is the list of values enclosed within braces **{ }** separated b a comma.

```
data_type array_name [size] = {value1, value2, ... valueN};
```


Array Initialization
Arr [5] = { 2, 4, 8, 12, 16 };
Memory Allocated and Initialized
Arr
2
4
8
12
16
0
1
2
3
4
Array Indexes

2. Array Initialization with Declaration without Size

If we initialize an array using an initializer list, we can skip declaring the size of the array as the compiler can automatically deduce the size of the array in these cases.

The size of the array in these cases is equal to the number of elements present in the initializer list as the compiler can automatically deduce the size of the array.

```
data_type array_name[] = {1,2,3,4,5};
```

The size of the above arrays is 5 which is automatically deduced by the compiler.

Example of Array Initialization in C

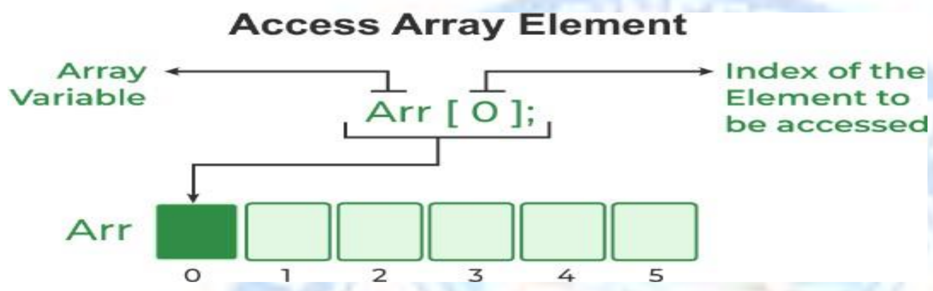
```
// C Program to demonstrate array initialization
#include <stdio.h>
int main()
{
   // array initialization using initialier list
   int arr[5] = { 10, 20, 30, 40, 50 };
   // array initialization using initializer list without specifying size
   int arr1[] = { 1, 2, 3, 4, 5 };
```

c) Accessing Array Elements

We can access any element of an array in C using the array subscript operator **[ ]** and the index value ***i*** of the element.

array_name [*index*];

One thing to note is that the indexing in the array always starts with 0, i.e., the **first element** is at index **0** and the **last element** is at **N – 1** where **N** is the number of elements in the array.



Example of Accessing Array Elements using Array Subscript Operator C

```
C Program to illustrate element access using array
// subscript
#include <stdio.h>
int main()
{
   // array declaration and initialization
   int arr[5] = { 15, 25, 35, 45, 55 };
   // accessing element at index 2 i.e 3rd element
   printf("Element at arr[2]: %d\n", arr[2]);
   // accessing element at index 4 i.e last element
   printf("Element at arr[4]: %d\n", arr[4]);
   // accessing element at index 0 i.e first element
   printf("Element at arr[0]: %d", arr[0]);
   return 0;
}
```

Output

```
Element at arr[2]: 35

Element at arr[4]: 55
Element at arr[0]: 15
```

d) <u>Update Array Element</u>

We can update the value of an element at the given index i in a similar way to accessing an element by using the array subscript operator **[ ]** and assignment operator **=**.

array_name[*i*] = *new_value*;

<u>Example :</u>

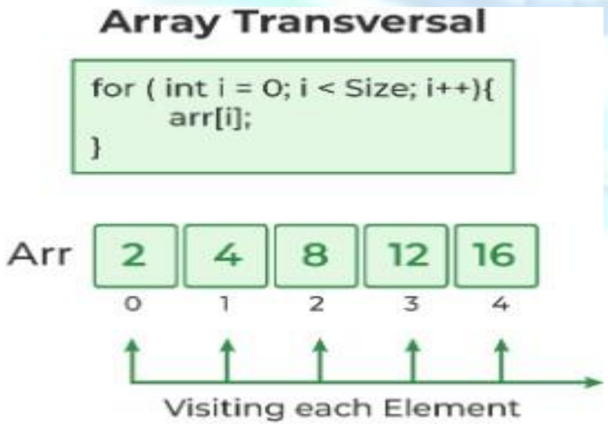
```
int arr[5] = {2, 4, 8, 16, 32};
arr[1]= 1;


// The new array will be {2, 1, 8, 16, 32}
```

e) <u>C Array Traversal</u>

Traversal is the process in which we visit every element of the data structure. For C array traversal, we use loops to iterate through each element of the array.

Array Traversal using for Loop

```
for (int i = 0; i < N; i++) {
    array_name[i];
}
```



How to use Array in C?

The following program demonstrates how to use an array in the C programming language:

```
// C Program to demonstrate the use of array
#include <stdio.h>
int main()
{
    // array declaration and initialization
    int arr[5] = { 10, 20, 30, 40, 50 };
    // modifying element at index 2
    arr[2] = 100;
    // traversing array using for loop
  printf("Elements in Array: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output

```
Elements in Array: 10 20 100 40 50
```