UNIT – 4 (Part-1)
Array

BCA SEM – 1

PROBLEM SOLVING METHODOLOGIS AND PROGRAMMING IN C

Code : CS-01

Presented by : Dhruvita Savaliya

Topics :

- Types of arrays
  - Single dimensional array
  - Two dimensional array
  - Multi-dimensional array
- String arrays
- Use of Arrays in Programming
- Arrays and Matrices

Use of array in Programming OR Why we need array in C?

- The fundamental data types, namely char int, float, and double are used to store only one value at any given time. Hence these fundamental data types can handle limited amounts of data.
- In some cases we need to handle large volume of data in terms of reading, processing and printing.
- To process such large amounts of data, we need a powerful data type that would facilitate efficient storing, accessing and manipulation of data items. For example: If the user wants to store marks of 100 students. This can be done by creating 100 variables individually but, this process is rather tedious and impracticable.
- This type of problem can be handled in C programming using arrays.

❖ Array :

- Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Array is Collection of similar datatype that stores in continues memory location.

(Or)

- An array is collection of homogeneous elements in a single variable.
- Array might be belonging to any of the data types
- Array size must be a constant value.
- Always, adjacent memory locations are used to store array elements in memory.
- Individual values are called as elements.
- It allocates sequential memory locations.

Types of Arrays :

- We can use arrays to represent not only simple lists of values but also tables of data in two or three or more dimensions.

- **One–dimensional arrays**
- **Two–dimensional arrays**
- **Multi-dimensional arrays**

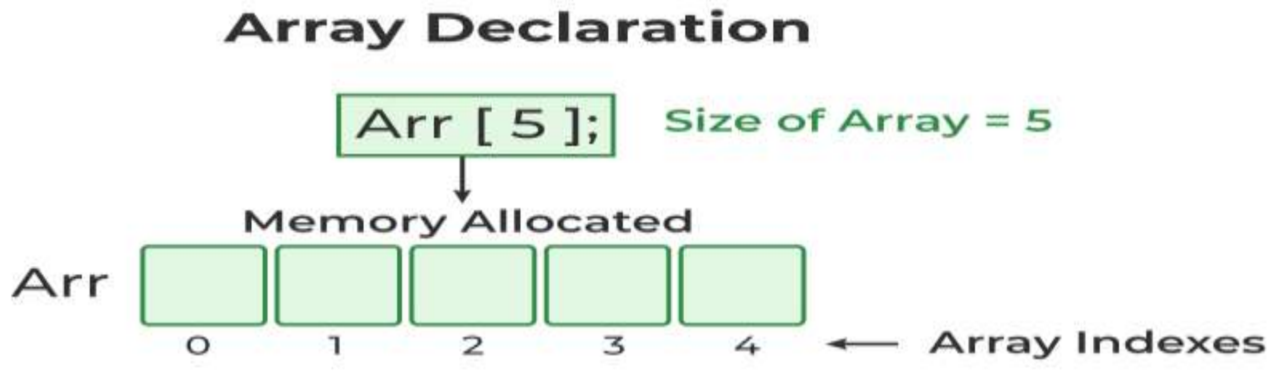
1. One - Dimensional Arrays:

- A list of items can be given one variable name using only **one subscript** and such a variable is called a single – subscripted variable or a one – dimensional array.
- It is also called **Vector.**
- **Declaration of One-Dimensional Arrays :**
- Like any other variables, arrays must be declared before they are used.
- **Syntax :**

<data type> <array name>[size_of_array];

- The data type specifies the type of element that will be contained in the array, such as int, float, or char.
- The size indicates the maximum number of elements that can be stored inside the array.
- For example :
- **int arr[5];**
- Here, the name of array is age. The size of array is 5, i.e., there are 5 items (elements) of array age. All elements in an array are of the same type (int, in this case).

- **Array elements :**
- Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program.

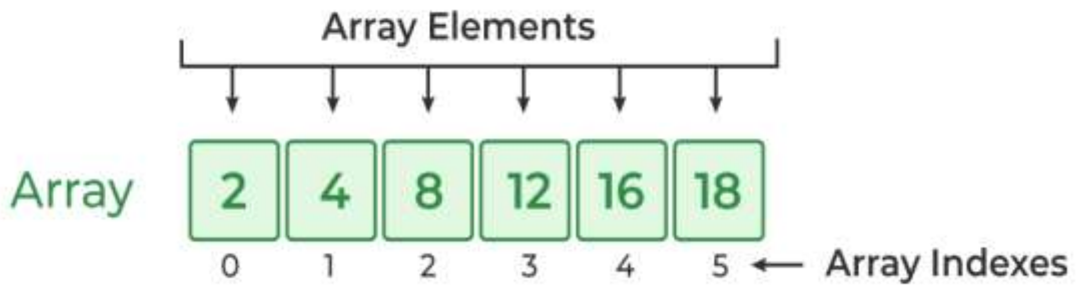


- **Note** that, the first element is numbered 0 and so on.(size-1)
- The first index is called **Lower Bound**, and the last index is called an **Upper Bound**. Upper Bound of a one dimensional is always Size – 1.
- Here, the size of array age is 5 times the size of int because there are 5 elements.
- Suppose, the starting address of age [0] is 2120d and the size of int be 2 bytes. Then, the next address (address of a [1]) will be 2122d, address of a [2] will be 2124d and so on.

- **<u>Initialization of one-dimensional array :</u>**
- After an array is declared, its elements must be initialized. Otherwise they will contain "garbage". We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
- The general form of initialization of array is .
- **<u>Syntax :</u>**
- **datatype array_name[size] = { list of values };**
- The values in the list are separated by commas.
- **You can initialize array in C either one by one or using a single statement as follows :**
- Array_name[0]=54;
- Array_name[1]=32;
- Array_name[2]=87;
- Array_name[3]=90;
- …….till size-1;

  Or
- int arr[5]={2,4,8,12,18};

- It is not necessary to define the size of arrays during initialization.
- int arr[ ]={2,4,8,12,16,18};
- In this case, the compiler determines the size of array by calculating the number of elements of an array.



Example :

```
#include<stdio.h>
void main()
{
    int a[5]={10,20,55,22,88},i;
    clrscr();
    for(i=0;i<5;i++)
    {
            printf("\n A[%d]%d",i,a[i]);
    }
    getch();
}
```

Output :

```
A[0]=10
A[1]=20
A[2]=55
A[3]=22
A[4]=88
```

2. Two - Dimensional Arrays :

- Two dimensional arrays have two Subscripts.
- The first Subscript denotes the number of rows and the second subscript denotes the number of columns.
- It is represent like a Table.
- Two dimensional array is also called matrix.
- **Syntax of 2D Array in C**

  array_name[size] [size];

  array_name[row_size] [column_size];

  Here,
- **size1:** Size of the first dimension.
- **size2:** Size of the second dimension.

- **Initializing Two- Dimensional Arrays :**
- • Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.
- **int table[2] [3] = {0,0,0,1,1,1};**
- • This initializes the elements of first row to zero and the second row to one.
- • This initialization is done row by row.
- • The above statement can be equivalently written as
- **int table[2][3] ={{0,0,0},{1,1,1}};**
- • we can also initialize a two-dimensional array in the form of a matrix as shown.
- **int table[2][3] = {**

```
                    {0,0,0},
                    {1,1,1}
               };
```

- Commas are required after each brace that closes of a row, except in case of last row.
- If the values are missing in an initialize, they are automatically set to zero.

Example :

```
# include<stdio.h>
 # include<conio.h>
 void main( ) {
        int  i,j;
        int  a[5][3] = {
                              {89,77,84},
                              {98,89,80},
                              {75,70,80},
                              {60,75,80},
                              {84,80,75}
                        };
        printf("elements of an array \n \n");
        for( i=0; i<5; i++)
        {
               for ( j=0; j<3; j++)
               {
                              printf ("%d\t", a[ i ][ j ]);
                }
               printf("\n");
         }
getch();
 }
```

	Column-0	column-1	column-2
	[0][0]	[0][1]	[0][2]
Row-0	89	77	84
	[1][0]	[1][1]	[1][2]
Row-1	98	89	80
	[2][0]	[2][1]	[2][2]
Row-2	75	70	80
	[3][0]	[3][1]	[3][2]
Row-3	60	75	80
	[4][0]	[4][1]	[4][2]
Row-4	84	80	75

3. Multi - Dimensional Arrays / Three Dimensional Array :

- A list of items can be given one variable name using **more than two subscripts** and such a variable is called Multi-dimensional array.
- Three Dimensional Arrays :
- The Three dimensional array can be declared as
- **Syntax :**
- <datatype> <array_name>[sizeofno.oftwoDimArray] [sizeofrow] [sizeofcolumn];

- **Initializing Three- Dimensional Arrays :**
- Like the one-dimensional arrays, three-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.
- **int table[2][2][3] = {0,0,0,1,1,1,6,6,6,7,7,7};**
- This initializes the elements of first two dimensional(matrix) first row to zero's and the second row to one's and second matrix elements are first row to six's and the second row to seven's.
- This initialization is done row by row.
- The above statement can be equivalently written as
- **int table[2][2][3] = {{{0,0,0},{1,1,1}},{{0,0,0},{1,1,1}}}**

- We can also initialize a two - dimensional array in the form of a matrix as shown.

```
int table[2][2][3] = {
    {
      {0,0,0},
        {1,1,1}
     },


      {
       {6,6,6},
         {7,7,7}
       }
   } ;
```

```
#include <stdio.h>
void main()
{
    int arr[2][2][2] = { 10, 20, 30, 40, 50, 60,70,80 };
    // printing elements
    for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                    for (int k = 0; k < 2; k++) {
                            printf("%d ", arr[i][j][k]);
                    }
                    printf("\n");
            }
            printf("\n \n");
    }
}
```

Output :

```
10 20
 30 40

 50 60
 70 80
```