Part-1:

- **Arrays:** Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example

Part-2:

- **Strings**: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions, Programming Example
- **Enumerated, Structure, and Union:** The Type Definition (Type def), Enumerated Types, Structure, Unions, and Programming Application.

Arrays

Why do we need ARRAYS?

Problem: When we have many data elements (10,20,30…n), we need many different variables (v1,v2,v3…vn). As the number of variables increases, the complexity of the program also increases

Solution: Arrays are used **to store multiple elements in a single variable (v[100]**, instead of declaring separate variables for each value.

Define an ARRAY:

Array is a collection of data elements with a similar data type. They are stored in the contiguous memory location. In the array, the first element is stored in index 0; the second element is stored in index 1, and so on. Arrays can be of a single dimension or multi-dimension.

An array is a special variable that is used to store multiple values of similar data types (homogeneous) at contiguous memory locations.

In C programming language, arrays are classified into two types. They are as follows:

- Single-Dimensional Array / One-Dimensional Array
- Multi-Dimensional Array

What are Single-Dimensional or One-Dimensional Arrays?

1. Description of Single-Dimensional Array
2. Declaration of Single Dimensional Array
3. Initialization of Single Dimensional Array
4. Accessing Elements of Single Dimensional Array
5. Example Application 1 of Single Dimensional Array - Sum & Average of n elements
6. Example Application 2 of Single Dimensional Array - Largest in n elements

1. Description of Single Dimensional Array:

Single-dimensional array or 1-D array is the simplest form of array in C. This type of array consists of elements of similar types and these elements can be accessed through their indices (positions).

2. Declaration of Single Dimensional Array:

In C programming language, when we want to create an array we must know

- the datatype of values to be stored in that array and
- also the number of values to be stored in that array.

Syntax1: Create 1-D array with Size:

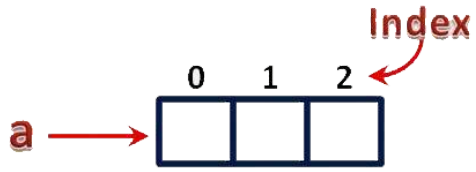
datatype arrayName [size] ;

- **datatype:** data type of array, Example: int, char, float, etc.
- **array_name:** Name of the array.
- **size:** Size of each dimension of the array

Example1: Declaration of 1-D Array with Size

```
// declare an array by specifying size in [].
int a[3];
```

Here, the compiler allocates 12 bytes of contiguous memory locations with a single name 'a' and tells the compiler to store three different integer values (each in 4 bytes of memory) into that 12 bytes of memory. For the above declaration, the memory is organized as follows.



All three memory locations in the above memory allocation have a common name 'a'. So accessing individual memory locations is not possible directly. Hence, the compiler assigns a numerical reference value to every individual memory location of an array. This reference number is called "Index" or "Subscript" or "Indices".

3. Declaration & Initialization of Single Dimensional Array:

Syntax2: Create 1-D array with Size and Initial values

datatype arrayName [size] = {value1, value2, ...} ;

Example2: Declaration of 1-D Array with Size and Initialization

```
// declare an array with size in [] and initial values.
int a[3] = {200, 100, 300};
```

In the above Syntax1 & Syntax2, the **datatype** specifies the type of values we store in that array and **size** specifies the maximum number of values that can be stored in that array.

Syntax3: Create 1-D array without Size and with Initial values

datatype arrayName [ ] = {value1, value2, ...} ;

Example3: Declaration of 1-D Array without Size and with Initialization

```
// declare an array with size in [] and initial values.
float salary[ ] = {1000.25, 2500.75, 3200.77, 500.97};
```

4. Accessing Elements of Single Dimensional Array:

The individual elements of an array are identified using the combination of **'arrayName'** and **'indexNumber'.** The Rules to access (to store or to retrieve) of Single or 1-D Array are,

- array name must be followed by an INDEX number of the element to be accessed.
- index value must be enclosed in square braces [ ].

- **index** value of an element in an array is the reference number given to each element at the time of memory allocation.
- index value of a 1-D array starts with zero (0) for the first element and increments by one for each element.
- index value in an array is also called a **subscript** or **indices**.

Syntax to access individual elements of a single dimensional array:

arrayName [indexNumber]

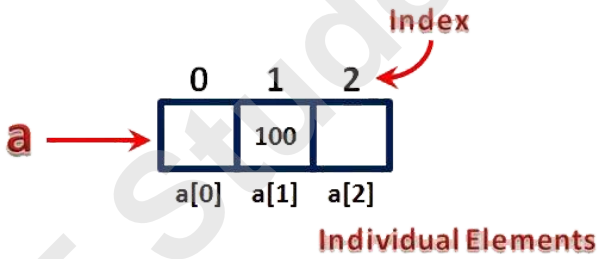
Example: Accessing 1-D array member

```
// declare an array with size 3
int a[3];
```

For this array "a", the individual elements can be denoted as follows. Assigns a value to the 2nd memory location.

a [1] = 100 ;

The result of the above assignment statement is as follows:



Note: The index of an array starts from 0 until it reaches the max (size – 1).

Example 1: [1-D array and "for" loop"]

```
/* Program to Print pre-initialized Array values
#include <stdio.h>
int main()
{   /* Array Declaration and also Initialization */
    int marks[10] = { 90, 91, 99, 93, 94};
    for (int i= 0; i < 5; i++)
    {
       printf("\n Element at position %d is %d",i, marks[i]);
    }
    printf("\n Element at 4th index is %d", marks[3]);
    return 0;
 }
```

OUTPUT:

Element at position 0 is 10
Element at position 1 is 91
Element at position 2 is 99
Element at position 3 is 93
Element at position 4 is 94

Element at 4th index is 93

5. Example Application 1 of Single Dimensional Array - Sum & Average of n elements

Example 2 : [1-D Array using for loop]

```
//  Program to Find Sum & Avg of n Elements using Loops and Variables
#include <stdio.h>
int main()
{    int n;
     int sum=0;
     float avg;

     printf("Enter size of the array: ");
     scanf("%d",&n);

     //Declaring array
     int arr[n];
     printf("Enter array elements\n");
      // Input array elements
     for(int i=0;i<n;i++)
     scanf("%d",&arr[i]);

     // Loop to find sum
     for(int i=0;i<n;i++)
          sum+=arr[i];
     printf("\nSum of the array is: %d",sum);
     avg = sum/n;
     printf("\nAverage of the array is: %.2f",avg);
     return 0;
}
```

Output:

Enter size of the array: 5

Enter array elements

50

100

75

100

80

Sum of the array is: 405

Average of the array is: 81.00

6. Example Application 2 of Single Dimensional Array - Largest of n elements

Example3: [1-D Array in for loop]

```
// Program to find the largest number in an array using loops
#include <stdio.h>
int main()
{   int size, i, largest;
    printf("\n Enter the size of the array: ");
    scanf("%d", &size);
    int array[size];  //Declaring array
    //Input array elements
    printf("\n Enter %d elements of the array: \n", size);
    for (i = 0; i < size; i++)
    {
        scanf(" %d", &array[i]);
    }
     //Declaring Largest element as the first element
    largest = array[0];
    for (i = 1; i < size; i++)
    {
        if (largest < array[i])
        largest = array[i];
    }
    printf("\n Largest element in the given array is: %d", largest);
    return 0;
 }
```

Output:

Enter the size of the array: 3

Enter 3 elements of the array:

90

155

45

Largest element in the given array is: 155

Multi-Dimensional Array

1. Description of Multi Dimensional Array
   a. 2-Dimensional
   b. 3-Dimensional
2. Declaration of Two-Dimensional Array
3. Initialization of Two-Dimensional Array
4. Accessing Elements of Two-Dimensional Array
5. Example Application 1 of Two-Dimensional Array
6. Example Application 2 of Two-Dimensional Array

1. Description of Multi-Dimensional Array:

An array of arrays is called a multi-dimensional array. An array created with more than one dimension or size is called a multi-dimensional array.

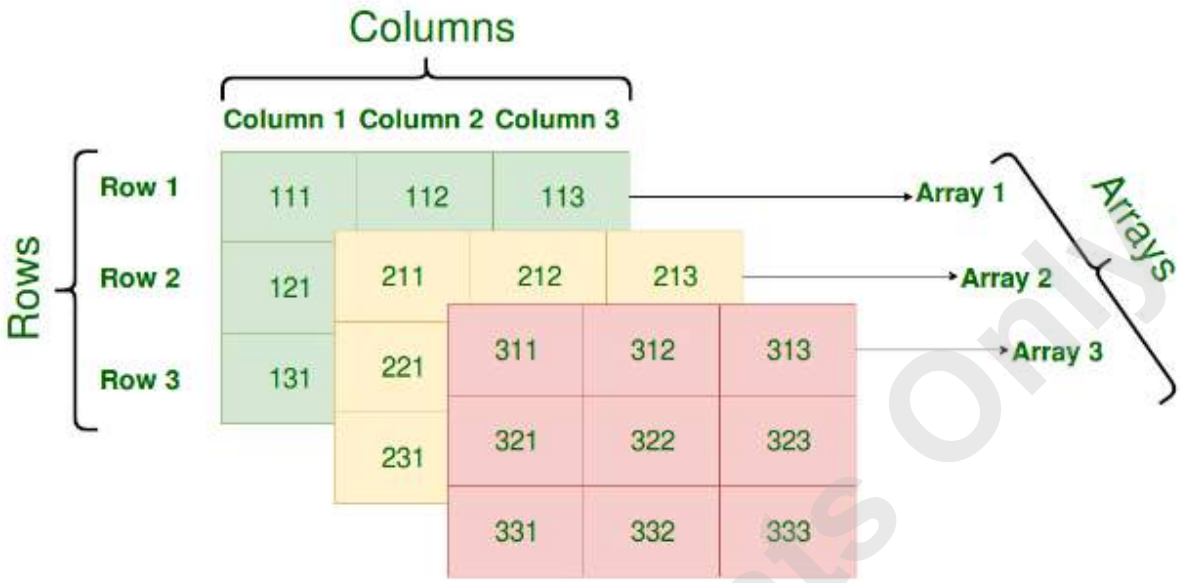
A multi-dimensional array can be a **two-dimensional array** or **three-dimensional array** or **four-dimensional array** or more.

2-D Array: arr[rows][cols]

	Col0	Col1	Col2
Row0	arr[0][0] 10	arr[0][1] 20	arr[0][2] 30
Row1	arr[1][0] 40	arr[1][1] 50	arr[1][2] 60

```
int arr[2][3] = { {10,20,30} . {40,50,60} };
```

3-D Array: y[arrays][rows][cols]



y[1][2][1] => 121 **y[2][1][2] => 212** **y[3][3][3] => 333**

The commonly used multi-dimensional array is a **two-dimensional array**. The 2-D arrays are used

- to store data in the form of a table with rows and columns,
- to create mathematical matrices,
- for drawing Chess boards,
- representing structures like a spreadsheet, etc.

2. Declaration of Two-Dimensional Array

Syntax for declaring a two-dimensional array

DataType arrayName [rowIndex] [columnIndex]

Example:

int matrix_A [2][3];

The above declaration of two-dimensional array reserves 12 continuous memory locations of 4 bytes each in the form of **2 rows** and **3 columns**.

3. Initialization of Two-Dimensional Array

Syntax for declaring and initializing a 2-D array with a specific number of rows and columns with initial values.

```
datatype arrayName [rows][colmns] = {
                                    {r1c1 value, r1c2 value, ...},
                                    {r2c1 value, r2c2 value, ...}
                                    …
                                   };
```

Example: Three Methods to Initialize an array (2 x 3 = 6 values)

Method1: First set will be the row1 and next set will be the row2

```
int matrix_A [2][3] = { {10, 20, 30},{40, 50, 60} };
(or)
int matrix_A [2][3] = {
                        {10, 20, 30},
                        {40, 50, 60}
                      };
```

Method2: First 3 values will be the row1 and next 3 values take row2

```
int matrix_A [2][3] = { 10, 20, 30, 40, 50, 60 };
```

Method3: User inputs data elements while running the program and saves in the array

```
int matrix_A[2][3];
for(int i = 0; i < 2; i++){
    for(int j = 0; j < 3; j++){
        scanf("%d",&matrix_A[i][j]);
    }
}
```

The above declaration methods of 2-D array reserves 6 contiguous memory locations of 4 bytes each in the form of 2 rows and 3 columns. And the first row is initialized with values 10, 20, 30 and the second row is initialized with values 40, 50, 60.

4. Accessing Individual Elements of Two-Dimensional Array

To access elements of a 2-D array in C, we use the 'arrayName' followed by the [rowIndex] and [columnIndex] of the element that needs to be accessed. Here the row and column index numbers must be enclosed in separate square braces. In the case of the two-dimensional array, the compiler assigns separate index values for rows and columns.

Syntax:

arrayName [rowIndex] [columnIndex]

Example:

matrix_A [0][1] = 10;

In the above statement, the element **10** will be saved at row index 0 and column index 1 of **matrix_A** array.

Note: For **1-D** array, we do not always need to specify the size. But for **2D** array, we must always specify the **column size**.

```
int arr[2][2] = {1, 2, 3,4 }      // Valid declaration
int arr[][2] = {1, 2, 3,4 }       // Valid declaration

// Invalid declaration - column dimension is compulsory
int arr[][] = {1, 2, 3,4 }
// Invalid declaration - column dimension is compulsory
int arr[2][] = {1, 2, 3,4 }
```

5. Example Application 1 of Two-Dimensional Array

```
//Program to print a 2D Array of elements already initialized
#include<stdio.h>
int main(void)
{
    // x array with 3 rows and 2 columns.
    int x[3][2] = {{10,11}, {12,13}, {14,15}};
```

```
    // display each array element
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            printf("Element at x[%i][%i]: ",i, j);
            printf("%d\n",x[i][j]);
        }
    }
     return (0);
}
```

Output:

Element at x[0][0]: 10
Element at x[0][1]: 11
Element at x[1][0]: 12
Element at x[1][1]: 13
Element at x[2][0]: 14
Element at x[2][1]: 15

6. Example Application 2 of Two-Dimensional Array - Read & Print of m x n size

```
//Program to read and print a 2D Array of m rows and n columns.
#include<stdio.h>
int main()
{//2D: Input number of rows x cols for 3 square arrays
int m,n;
int arr2d[30][30];
printf("\n Enter Number of rows cols for square array: ");
scanf("%d %d",&m,&n);
for(int i=0;i<m;i++)
{
    for(int j=0;j<n;j++)
    {
        printf("Value [%d,%d]: ",i,j);
        scanf("%d",&arr2d[i][j]);
    }
}
```

```
//2D: print m x n values
printf("\n 2D array is \n");
for(int i=0; i<m; i++)
{
    for(int j=0;j<n;j++)
    {
        printf(" %d ",arr2d[i][j]);
    }
    printf("\n");
}
return 0;
}
```

OUTPUT: `Enter Number of rows cols for square array: 2 2`

Value [0,0]: 10

Value [0,1]: 20

Value [1,0]: 30

Value [1,1]: 40

2D array is

10 20

30 40

7. **Example Application 3 of Two-Dimensional Array - Add two 2x2 arrays and save in third array and print the result.**

```
//Arrays 2D: Add two square arrays and svae the result in third array
#include<stdio.h>
int main()
{
//2D: read row1xcol1 values
int rows,cols;
int a1[30][30];
int a2[30][30];
int a3[30][30];
printf("\n Enter rows cols for Arrays a1 and a2: ");
scanf("%d %d",&rows,&cols);

//Read 2D Array1 values
printf("\n Input values for Array1: \n");
```

```
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf("Value [%d,%d]: ",i,j);
        scanf("%d",&a1[i][j]);
    }
}
//Read 2D Array2 values
printf("\n Input values for Array2: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf("Value [%d,%d]: ",i,j);
        scanf("%d",&a2[i][j]);
    }
}
//2D: print Array1
printf("\n Array1: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf(" %d ",a1[i][j]);
    }
    printf("\n");
}
//2D: print Array2
printf("\n Array2: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf(" %d ",a2[i][j]);
    }
    printf("\n");
}
```

```
//2D: Add 2 arrays and save into 3rd array
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        a3[i][j] = a1[i][j] + a2[i][j];
    }
}
//2D: print Array3 with added values
printf("\n Added values in Array3: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf(" %d ",a3[i][j]);
    }
    printf("\n");
}
return 0;
}
```

Output:

Enter rows cols for Arrays a1 and a2: 2 2
Input values for Array1:
Value [0,0]: 10
Value [0,1]: 20
Value [1,0]: 30
Value [1,1]: 40

Input values for Array2:
Value [0,0]: 1
Value [0,1]: 2
Value [1,0]: 3
Value [1,1]: 4

Array1:
10 20
30 40

Array2:
1 2
3 4

Added values in Array3:
11 22
33 44

Advantages of Array in C

Arrays have a great significance in the C language.

- Arrays make the program optimized and clean
- We can store multiple elements in a single array at once; so, we do not have to write or initialize them multiple times.
- Every element can be traversed in an array using a single loop statement.
- Easier to sort data elements with a few lines of code.
- Any array element can be accessed in any order either from the front or rear in O(1) time.

Applications of Arrays in C

In C, arrays are used in a wide range of applications.

● **Arrays are used to Store List of values -** Single dimensional arrays are used to store a list of values of the same datatype in a row or in a linear form.

● **Arrays are used to Perform Matrix Operations -** Two-dimensional arrays are used to create matrices. We can perform various operations on matrices using two-dimensional arrays.

● **Arrays are used to implement Search Algorithms -** We use single-dimensional arrays to implement search algorithms such as

1. Linear Search
2. Binary Search

● **Arrays are used to implement Sorting Algorithms -** We use Single dimensional arrays to implement sorting algorithms such as,

1. Insertion Sort
2. Bubble Sort
3. Selection Sort
4. Quick Sort
5. Merge Sort, etc.,

● **Arrays are used to implement Datastructures -** We use single dimensional arrays to implement data structures such as

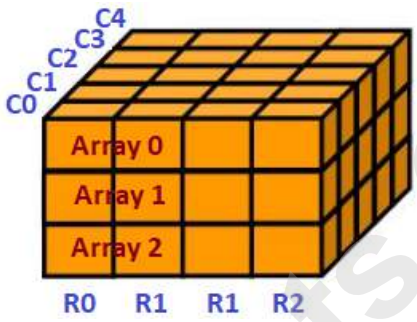
1. **Stack Using Arrays**
2. **Queue Using Arrays**

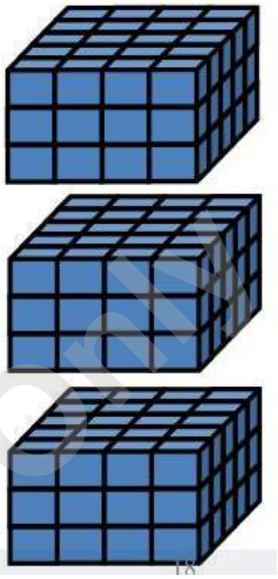
● **Arrays are also used to implement CPU Scheduling Algorithms**

Visual Representation of Single-Dimensional and Multi-Dimensional Arrays

	Col0	Col1	Col2
Row0	arr[0][0] 10	arr[0][1] 20	arr[0][2] 30
Row1	arr[1][0] 40	arr[1][1] 50	arr[1][2] 60

int arr[2][3] = { {10,20,30} . {40,50,60} };



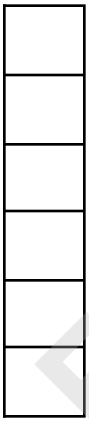


2-D Array
A[3][4]
3 Rows
4 Columns

3-D Array
B[3][4][5]
3 Arrays
4 Rows
5 Columns

4-D Array
C[3][4][5][3]
3 Arrays
4 Rows
5 Columns
3 3-D Cubes

1-D Array X[6]



1D Array Y[6]

*** Important Characteristics of Arrays *** [EXAM Bits]	

1	An array address is the address of the first element of the array itself.
	Ex: **int arr[3] = {100, 200, 300};**
	"arr" is the name of the array; it does not refer to any value
	"arr" points to the memory address of 1st element
	"arr" refers to the address that is same of "&arr[0]"
	Ex:
	`#include<stdio.h>`
	`int main(void)`
	`{ int arr[3] = {100, 200, 300};`
	`printf("Mem Address of array : %p \n", arr);`
	`printf("Mem Address of 1st element: %p", &arr[0]);`
	`return 0;`
	`}`
	Output:
	`Mem Address of array : 0061FF18`
	`Mem Address of 1st element: 0061FF18`

2	If you do not initialize an array, you must mention ARRAY SIZE. Incorrect declaration: **int arr[ ];** Correct declaration: **int arr[ ] = {5, 15, 25, 35};** **Note:** You can skip the SIZE of an array if you initialize with values.
3	Array size is the sum of the sizes of all elements of the array. Ex: **float salaries[10];** //assuming one float value size is 4 bytes The total size of the "salaries" array will be: **40bytes** (4bytes x 10 elements)
4	Types of Arrays: int, long, float, double, struct, enum, or char All elements in one array must be of the same data type. Ex: **char grade[5] = { 'A', 'B', 'C', 'D', 'F' };**
7.	An array's index always starts with 0.
6	An array size can not be changed once it is created.
7	The value (element) in an Array can be changed any number of times. Ex: **int a[10] = {10, 20, 30};** **a[1] = 15;** //this changes the 2nd element 20 to 15. Now, the array 'a' will have **10, 15, 20** elements
8	To access Nth element of an array "customers", use customers[n-1] because the starting index is 0.
9	arr[i] and i[arr], both notations refer to the same array element. Ex: `char arr[4] = { 'A', 'B', 'C', 'F' };`

```
        int i = 0;
        while (i<3) {
           printf("%c ", arr[i]);
           printf("%c", i[arr]);
           printf("\n");
        }
```

Output:

A A
B B
C C
F F

Note:

For Part-2, Refer to PTC UNIT III, Part-2 document

Part-2:

- **Strings**: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions, Programming Example
- **Enumerated, Structure, and Union:** The Type Definition (Type def), Enumerated Types, Structure, Unions, and Programming Application.