1. Introduction to Arrays

An **array** in C language is a **collection of elements of the same data type**, stored in **contiguous memory locations** and accessed using a **common name**.

Arrays are used when we need to store and process **multiple values of similar type** efficiently.

Example (Without Array)

```
int a, b, c, d, e;
```

Example (With Array)

```
int a[5];
```

---

2. Need for Arrays

Arrays are required to:

- Store large amounts of data efficiently
- Reduce number of variables
- Simplify program logic
- Process data using loops
- Improve memory management

---

3. Declaration of Arrays

Syntax

```
data_type array_name[size];
```

Example

```
int marks[10];
float price[5];
```

Important Points

- Size must be constant
- Index starts from **0**
- Last index is **size – 1**

---

4. Initialization of Arrays

4.1 Compile-Time Initialization

```
int a[5] = {10, 20, 30, 40, 50};
```

If size is not mentioned:

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

4.2 Run-Time Initialization

```
int i;
for(i = 0; i < 5; i++)
{
   scanf("%d", &a[i]);
}
```

---

5. Accessing Array Elements

Array elements are accessed using **index number**.

Example

```
printf("%d", a[2]);
```

---

6. One-Dimensional Array (1D Array)

A **one-dimensional array** stores data in a **linear form**.

Example

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

Program: Sum of Elements

```
int sum = 0, i;
for(i = 0; i < 5; i++)
   sum += arr[i];
```

---

7. Two-Dimensional Array (2D Array)

A **two-dimensional array** is used to store data in **tabular form (rows and columns)**.

Syntax

```
data_type array_name[rows][columns];
```

Example

```
int mat[3][3];
```

Initialization

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

8. Accessing 2D Array Elements

```
printf("%d", mat[1][0]);
```

Nested Loop Example

```
int i, j;
for(i = 0; i < 2; i++)
{
   for(j = 0; j < 2; j++)
      printf("%d ", mat[i][j]);
}
```

---

9. Multidimensional Arrays

Arrays with more than two dimensions are called **multidimensional arrays**.

Example

```
int arr[2][3][4];
```

Used in:

- Scientific calculations
- Image processing

---

10. Array and Functions

Arrays can be passed to functions.

Example

```
void display(int a[], int n)
{
   int i;
   for(i = 0; i < n; i++)
      printf("%d ", a[i]);
}
```

---

11. Passing 2D Arrays to Functions

```
void display(int a[2][2])
{
   int i, j;
   for(i = 0; i < 2; i++)
      for(j = 0; j < 2; j++)
         printf("%d ", a[i][j]);
```

```
}
```

---

12. Array and Pointers

Array name stores the **base address** of the array.

Example

```
int a[5];
int *p = a;
```

Access using pointer:

```
*(p + 1)
```

---

13. Searching in Arrays

13.1 Linear Search

```
for(i = 0; i < n; i++)
{
   if(arr[i] == key)
   {
      printf("Found");
      break;
   }
}
```

---

14. Sorting Arrays

14.1 Bubble Sort

```
for(i = 0; i < n - 1; i++)
{
   for(j = 0; j < n - i - 1; j++)
   {
      if(arr[j] > arr[j + 1])
      {
         temp = arr[j];
         arr[j] = arr[j + 1];
         arr[j + 1] = temp;
      }
   }
}
```

---

15. Advantages of Arrays

- Efficient data storage
- Easy data access
- Compatible with loops
- Reduces code size

---

16. Limitations of Arrays

- Fixed size
- Stores only same data type
- Insertion and deletion difficult
- Memory wastage possible

---

17. Common Errors in Arrays

1. Accessing out-of-bound index
2. Wrong size declaration
3. Uninitialized array
4. Confusion in multidimensional indexing

---

18. Difference Between Array and Variable

Feature	Variable	Array
Stores values	One	Multiple
Memory	Single	Contiguous
Access	Direct	Index-based

---

19. Applications of Arrays

- Data processing
- Image and signal processing
- Sorting and searching
- Matrix operations
- Scientific computing

20. Conclusion

Arrays are one of the most important data structures in C language. They allow efficient storage and manipulation of multiple data values. Understanding arrays is essential for advanced topics like **strings, pointers, structures, and data structures**.