1. Introduction to Arrays

An array in C++ is a collection of elements of the **same data type** stored in **contiguous memory locations**. Arrays allow storing multiple values under a single variable name, which makes programs more organized and efficient.

Instead of declaring multiple variables for similar data, arrays enable us to store and process large amounts of data easily. Arrays are widely used in almost every C++ program, especially where data repetition is required.

---

2. Need for Arrays

Without arrays, programmers would need to declare separate variables for each value, which becomes impractical when dealing with large data sets.

Arrays help in:

- Storing large amounts of data
- Reducing code length
- Easy data processing
- Efficient memory management
- Implementing searching and sorting algorithms

---

3. Characteristics of Arrays

Key characteristics of arrays in C++:

- All elements must be of the same data type
- Elements are stored in consecutive memory locations
- Array index starts from **0**
- Fixed size (cannot be changed during execution)
- Fast access to elements using index

---

4. Declaration of Arrays

An array must be declared before use.

Syntax

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

Example

```
int marks[5];
```

This creates an array named marks that can store 5 integer values.

---

5. Initialization of Arrays

Arrays can be initialized at the time of declaration.

Example

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

If the size is not specified, the compiler automatically determines it.

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

---

6. Accessing Array Elements

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

Syntax

```
array_name[index];
```

Example

```
cout << numbers[0];
```

This prints the first element of the array.

---

7. Input and Output Using Arrays

Input Example

```
for (int i = 0; i < 5; i++)
{
   cin >> marks[i];
}
```

Output Example

```
for (int i = 0; i < 5; i++)
{
   cout << marks[i] << endl;
}
```

---

8. One-Dimensional Arrays

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

Example

```
int arr[3] = {5, 10, 15};
```

Uses

- Storing marks
- Storing prices
- List of items

---

9. Two-Dimensional Arrays

Two-dimensional arrays store data in **row and column** format.

Syntax

```
data_type array_name[row][column];
```

Example

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

---

10. Input and Output of 2D Arrays

Input

```
for(int i = 0; i < 2; i++)
{
   for(int j = 0; j < 2; j++)
   {
      cin >> matrix[i][j];
   }
}
```

Output

```
for(int i = 0; i < 2; i++)
{
   for(int j = 0; j < 2; j++)
   {
      cout << matrix[i][j] << " ";
   }
   cout << endl;
}
```

---

11. Memory Representation of Arrays

Array elements are stored in contiguous memory locations.

Address formula:

Address = Base Address + (Index × Size of Data Type)

This allows fast access to any element.

---

12. Passing Arrays to Functions

Arrays can be passed to functions as arguments.

Example

```
void display(int arr[], int size)
{
   for(int i = 0; i < size; i++)
      cout << arr[i];
}
```

---

13. Advantages of Arrays

- Easy data storage
- Fast access
- Efficient looping
- Useful in algorithms
- Reduced code complexity

---

14. Limitations of Arrays

- Fixed size
- Cannot store different data types
- Memory wastage if size is large
- No built-in bounds checking

---

15. Common Errors in Arrays

- Accessing out-of-bound index
- Using uninitialized arrays
- Incorrect loop conditions
- Wrong array size

---

16. Applications of Arrays

Arrays are used in:

- Sorting algorithms
- Searching algorithms
- Matrix operations
- Data analysis

- Game programming

---

17. Difference Between Array and Variables

Variable	Array
Stores one value	Stores multiple values
Simple memory use	Structured memory use
Less efficient for large data	Efficient for large data

---

18. Best Practices for Using Arrays

- Use correct size
- Initialize arrays
- Use loops properly
- Avoid magic numbers
- Validate index values

---

19. Arrays vs Vectors (Brief Idea)

Arrays have fixed size, while vectors are dynamic in nature. Vectors are safer and more flexible but arrays are faster and simpler.

---

20. Conclusion

Arrays are a fundamental data structure in C++. They allow storing and processing multiple values efficiently. Understanding arrays is essential for mastering advanced topics like strings, structures, pointers, and dynamic memory allocation.