1.4 ARRAYS

Definition

An array is a collection of elements of the **same data type** stored in **contiguous memory locations** and accessed using a **common name** along with an **index or subscript**.

Declaration of Arrays

Declaring an array requires specifying the data type, array name, and size. The size determines how many elements the array can store.

Syntax

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

Example

```
int numbers[10];
float salary[5];
char name[20];
```

Features of Arrays:

1. All elements in an array must be of the same type (e.g., all int, all float).
2. **Common name** refer to the array by a single identifier (e.g., a).
3. Each element is accessed by its position, called an index or subscript, usually starting from 0 in most programming languages.
4. Memory allocation is done at **compile time** (for static arrays).
5. Elements are stored one after another in **continuous** memory, which allows efficient access.
6. Fast access using the **indexing operator** ([]).

Example:

int a[5]; → creates an integer array of size 5.

1.4.1 Types of Arrays

a) One-Dimensional Array

Stores data in a single row (linear structure). A one-dimensional array represents a simple list of values arranged in a single row. It is the simplest form of an array and is commonly used for storing lists like marks, salaries, or temperatures.

Example:

int a[5];

b) Two-Dimensional Array

A two-dimensional array stores data in the form of rows and columns, similar to a matrix. This type of array is useful for applications such as tables, grids, and mathematical matrix operations. Represented as rows and columns (matrix format).

Example:

int a[3][3];

c) Multi-Dimensional Array

Array with more than two dimensions. A multi-dimensional array extends the concept of a matrix into three or more dimensions, enabling representation of complex structures such as 3D data blocks or multi-level tables.

Example:

int a[3][3][3];

Initialization of Arrays

a) Compile-Time Initialization

Arrays can be initialized during declaration by assigning values inside braces. If

all values are specified, the array gets fully initialized

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

b) Partial Initialization

If only some values are provided, the remaining elements automatically receive zero as their value. Remaining values become **0**.

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

c)Automatic Size Calculation

C++ also allows automatic size calculation when the number of values is known, in which case the compiler determines the appropriate array size.

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

Accessing Array Elements

Each element of an array is accessed using its index inside square brackets. Indexing begins at zero, meaning the first element is accessed using a[0].

Elements are accessed by index:

a[0], a[1], a[2] … a[n-1]

Example:

```
cout << a[2];
```

Input and Output of Array Elements

Input

```
for(i=0; i<5; i++)
    cin >> a[i];
```

Output

```
for(i=0; i<5; i++)
    cout << a[i] << " ";
```

2D Array – Declaration & Initialization

A two-dimensional array is declared by providing two sizes: one for rows and another for columns. It can be initialized using nested braces, where each inner brace represents a row of the matrix. Elements are accessed using two indices—one specifying the row and the other specifying the column.

Declaration

```
int m[3][3];
```

Initialization

```
int m[2][3] = {
                {1, 2, 3},
                {4, 5, 6}
               };
```

Accessing 2D elements

```
cout << m[0][1];   // prints 2
```

Memory Representation

- Each element occupies memory based on its type (e.g., int = 4 bytes).

- Index calculation:

  Address of a[i] = base_address + (i * size_of_each_element)

Example (1D Array)

```
#include <iostream.h>
#include <conio.h>

class ArrayDemo
{
public:
   int a[5];                                  // Array as a class member

   void read()                    // Member function to read array elements
   {
      cout << "Enter 5 elements:\n";
      for (int i = 0; i < 5; i++)
         cin >> a[i];
   }

   void display()              // Member function to display array elements
   {
      for (int i = 0; i < 5; i++)
         cout << a[i] << " ";
   }
};

void main()
{
   clrscr();
   ArrayDemo d;            // Object name starts with first letter of class → d
   d.read();                                  // Read array elements from user
```

```
    d.display();                    // Display array elements
    getch();
}
```

Output:-

```
Enter 5 elements:
10 20 30 40 50
10 20 30 40 50
```

- ✓ The array a[5] is declared as a data member of the class ArrayDemo.
- ✓ An object d of class ArrayDemo is created.
- ✓ d.read() is called to **accept array elements from the user**.
- ✓ The display() function is a member function that prints all array elements.
- ✓ An object d is created, and the function is called using the object to show array values.
- ✓ Thus, the program shows how **arrays can be used inside a class** and how **member functions can access and manipulate class data**.

Advantages of Arrays

- Arrays allow storing multiple values of the same data type in an organized way.
- They provide fast and direct access to elements using their index.
- Arrays make operations like sorting and searching more efficient.
- Their contiguous memory allocation improves cache performance and speeds up execution.

Disadvantages of Arrays

- Arrays have a fixed size, so they cannot grow or shrink during program execution.
- Memory may be wasted if the declared array size is larger than the actual data stored.

- Inserting or deleting elements is slow because it requires shifting other elements.
- Arrays cannot store elements of different data types together.

Applications of Arrays

1. **Storing Collections of Data**

   Arrays are used to store multiple values of the same type under a single name., such as student marks, employee salaries, or daily temperatures.
2. **Matrix Representation in Mathematics**

   Arrays (especially 2D arrays) are used to represent matrices for mathematical, engineering, and scientific calculations.
3. **Lookup Tables**

   Arrays store fixed sets of values that can be accessed quickly, helping in fast search operations and decision-making in programs.
4. **Image Storage and Processing**

   Digital images are represented using 2D arrays, where each element represents a pixel value (color or intensity).
5. **String Handling**

   Character arrays are used to store strings in languages like C, where each character is stored in a continuous memory block.
6. **Data Sorting and Searching**

   Arrays are commonly used to apply algorithms like bubble sort, binary search, and selection sort.
7. **Implementing Other Data Structures**

   Arrays form the base for many advanced data structures such as stacks, queues, heaps, and hash tables.