Unit 4
Arrays, Pointers & Structures in C

ARRAYS

BASIC CONCEPTS OF ARRAYS

An array is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow multiple values to be stored using a single variable name. Each element is accessed using an index (starting from 0).
Array declaration syntax:

$$\boxed{data\_type\ array\_name[size];}$$

Example: $int\ numbers[10];$ declares an array of 10 integers.
Why Use Arrays?

- To store large sets of related data like marks, salaries, temperatures, etc.
- Efficient access using index.
- Used in looping, sorting, searching, and matrices.

ONE-DIMENSIONAL ARRAYS

- Linear collection of elements in a single row.
  Syntax:
  $$\boxed{data\_type\ array\_name[size];}$$
  Example: $float\ prices[5];$
- Accessing elements: $prices[0], prices[1], \ldots, prices[4]$
- Initialization at declaration:
  $$\boxed{int\ a[3]\ =\ \{10, 20, 30\};}$$
- Looping over 1D array:
  $$\boxed{\begin{array}{l} for(i\ =\ 0;\ i\ <\ size;\ i++) \\ printf("\%d", a[i]); \end{array}}$$

TWO-DIMENSIONAL ARRAYS

- Represents a matrix (rows and columns).
  Syntax: $data\_type\ array\_name[rows][cols];$
  Example: $int\ matrix[2][3];$
- Initialization:
  $$\boxed{\begin{array}{l} int\ matrix[2][3]\ =\ \{ \\ \quad 1 \quad 2 \quad 3 \\ \quad 4 \quad 5 \quad 6 \\ \}; \end{array}}$$
- Access: $matrix[0][1]$ refers to 2.

MULTIDIMENSIONAL ARRAYS

- Arrays with more than 2 dimensions.
- Commonly used in scientific and graphical computations.
  Syntax:
  $$\boxed{data\_type\ array\_name[d1][d2][d3]\ldots;}$$
  Example: $int\ arr[2][2][3];$
- Complex but stored linearly in memory.

C PROGRAMMING EXAMPLES RELATED TO ARRAYS

- Array traversal (looping)
- Array sum, max/min
- Searching and sorting arrays (Bubble sort, Linear search)
- Matrix multiplication using 2D arrays

POINTERS

Pointer Basics

- A pointer is a variable that stores the address of another variable.
  Syntax: $data\_type\ *pointer\_name;$
  Example:
  $$\boxed{\begin{array}{l} int\ a\ =\ 10; \\ int\ *p\ =\ \&a; \end{array}}$$

Why Use Pointers?

- Efficient handling of arrays, strings
- For dynamic memory allocation
- Pass by reference in functions
- Creating linked lists, trees, and graphs

POINTER ARITHMETIC

- You can perform operations: $+, -, ++, --$ on pointers
  Example:
  $p++;$ // Moves to next memory location
- Depends on data type (e.g., int advances by 4 bytes)

PASSING ARRAYS USING POINTERS

- Arrays are passed as pointers to functions.
- Only the address of the first element is passed.
  Example:
  $$\boxed{void\ display(int\ *arr, int\ size);}$$

SIZE OF POINTER

- Independent of data type, but depends on architecture:
  - 32-bit system → 4 bytes
  - 64-bit system → 8 bytes

MEMORY ALLOCATION FUNCTIONS

- Used for Dynamic Memory Allocation (DMA) at runtime.
- Provided in $< stdlib.h >$

Function	Description
malloc()	Allocates uninitialized memory
calloc()	Allocates and initializes memory
realloc()	Resizes previously allocated memory
free()	Frees the allocated memory

ARRAYS OF POINTERS

- Array that stores addresses.
  Example:

```
int *arr[5];
```

- Useful for strings, function pointers.

POINTERS TO VOID (VOID POINTERS)

- Can store the address of any data type.
  Syntax: $void\ *ptr;$
- Requires explicit casting to dereference.

COMMAND-LINE ARGUMENTS

- Allow users to pass values to $main()$ when program starts.
  Syntax:

```
int main(int argc, char *argv[])
```

- $argc$: Argument count
- $argv[]$: Argument vector (array of strings)
- Useful in file handling, automation, scripting

STRUCTURES IN C

Definition

Structure is a user-defined data type that allows grouping different data types together.

Syntax:

```
struct student {
int id;
char name[20];
float marks;
}
```

Accessing Members

- Via structure variable using dot (.) operator:
- $s1.id\ =\ 101;$

Uses of Structures

- Used in:
  - Records (students, employees)
  - File handling
  - Complex data models (3D points, time, etc.)

UNIONS IN C

Definition

Similar to structures, but shares memory among all members.

Syntax:

```
union data {
int i;
float f;
char c;
};
```

KEY DIFFERENCE (STRUCTURE VS UNION)

Feature	Structure	Union
Memory	Sum of all fields	Max size of one field
Access	All members	One member at a time
Use Case	All data needed	One value at a time

ENUMERATION

Definition

- enum is a user-defined data type consisting of named integer constants.
  Syntax:

  ```
  enum color { RED, GREEN, BLUE };
  ```

- By default, values start from 0.

Benefits of Enum

- Improves readability
- Makes code more maintainable
- Prevents use of magic numbers