B.Sc. (I.T.) / M.Sc. (I.T.) 1s Semester

Course: 104: Fundamentals of Programming Using C-1

4.1 One-Dimensional Array

A **one-dimensional array (1D array)** stores a list of elements **of the same data type** in **continuous memory locations**.
This makes searching, sorting, and processing extremely fast.

Why 1D Arrays Are Needed
Suppose we want to store marks of 50 students.
Using 50 variables is impossible:
int m1, m2, m3, ..., m50;

Instead:
int marks[50];

Advantages:
✓ Memory-efficient
✓ Easy to process using loops
✓ Fast access using index
✓ Predictable memory layout

4.2 Declaration & Initialization of Array

Declaration
data_type array_name[size];

Examples:
int marks[5];
float prices[10];
char vowels[5];

Logic Behind Indexing

Indexes start from 0 because:

Memory offset calculated as:
address = base_address + (index * size_of_data_type)

So index 0 naturally refers to the base address.

Initialization
1. Complete Initialization
int a[5] = {10, 20, 30, 40, 50};

2. Partial Initialization (rest become 0)
int a[5] = {1, 2};   // → {1,2,0,0,0}

3. Automatic Size
int a[] = {5, 10, 15};

4. Runtime Input
```
for(i=0;i<5;i++)
   scanf("%d",&a[i]);
```

Logic During Runtime Input

Each scanf reads individual elements and stores them in consecutive memory blocks.

4.3 Two-Dimensional Array (Matrix)

A 2D array is like a table of rows and columns.

Example:
int a[3][3];

Represents:
[a00 a01 a02]
[a10 a11 a12]
[a20 a21 a22]

4.3.1 Declaration

int matrix[rows][columns];
Example:
int a[4][3];   // 4 rows, 3 columns

4.3.2 Accessing Matrix Elements

a[row][column]
Logic Behind Accessing Element
For element a[i][j]:
address = base + ((i * total_columns) + j) * size
This is called **row-major order** (C language default).

Example
Input for 2×2 matrix:
```
for(i=0;i<2;i++)
  for(j=0;j<2;j++)
    scanf("%d",&a[i][j]);
```
Here:
- Outer loop changes row
- Inner loop fills each column of that row
  This ensures predictable tabular input.

4.3.3 Operations on Matrix

1. Matrix Addition
c[i][j] = a[i][j] + b[i][j];

Logic
Each element in the resulting matrix is sum of elements at **same position**.
Example:
A = [1 2]
[3 4]

B = [5 6]
[7 8]

C = [1+5 2+6]
[3+7 4+8]

2. Matrix Subtraction
c[i][j] = a[i][j] - b[i][j];

Logic
Position-wise subtraction.

3. Matrix Multiplication
Formula:
C[i][j] = sum (A[i][k] * B[k][j])

Logic
- Take **row** from A
- Take **column** from B
- Multiply corresponding elements
- Add all products

Example Step-by-Step:
A (2×3) and B (3×2)
A B
[1 2 3] [1 2]
[4 5 6] [3 4]
[5 6]

Compute C[0][0]:
(1×1) + (2×3) + (3×5)
= 1 + 6 + 15
= 22
This logic applies for all elements.

4. Diagonal Elements

Main diagonal
a[i][i]
Secondary diagonal
a[i][n-i-1]
Why?
Because secondary diagonal moves backward across columns.

5. Row & Column Sum

Row Sum Logic
Fix row
Vary column
Column Sum Logic
Fix column
Vary row

6. Transpose

t[j][i] = a[i][j];
Logic
Row becomes column → column becomes row.

4.4 Array Manipulation (With Logic)

These operations help manage data inside arrays.

4.4.1 Searching

1. Linear Search

Algorithm:

1. Start from index 0
2. Compare each element with key
3. Stop if found

Logic

Sequential check until match found.

Example:

```
for(i=0;i<n;i++)
  if(a[i] == key)
    found = i;
```

Time Complexity: **O(n)**

2. Binary Search

Works only on sorted arrays.

Logic:

1. Find middle element
2. Compare key with mid
3. If key < mid → search left
4. Else → search right

This halves the array each time.

Time Complexity: **O(log n)**

4.4.2 Insertion

To insert element at position pos:

1. Start from last element
2. Shift all elements **right**
3. Insert value at pos

Example:

```
for(i=n-1;i>=pos;i--)
   a[i+1] = a[i];

a[pos] = value;
```

Logic

Shifting is necessary because arrays have fixed contiguous storage, so you cannot "create space" without shifting.

4.4.3 Deletion

Algorithm:
1. Start from position pos
2. Shift elements **left**
3. Overwrite the element to be deleted

Example:

```
for(i=pos;i<n;i++)
    a[i] = a[i+1];
```

Logic
Left shifting closes the gap created after deleting.

4.4.4 Modification
Changing a specific element.
a[pos] = new_value;
Logic
Direct access by index makes modification **O(1)** constant time.

4.4.5 Sorting

A. Bubble Sort
Compares adjacent elements and swaps if needed.
Logic:
- Bubbles larger element to end in each pass.

Example (Step-by-step for [5,3,1]):

Pass 1:
5 > 3 → swap → [3,5,1]
5 > 1 → swap → [3,1,5]

Pass 2:
3 > 1 → swap → [1,3,5]
Final: [1,3,5]

B. Selection Sort
- Find smallest element
- Put at correct position

Logic:
- Select minimum
- Place at front
- Continue for rest

C. Insertion Sort

- Consider first element sorted
- Insert next element in sorted part

Logic:
Similar to playing cards sorting in a hand.

4.5 Multidimensional Array

Arrays with more than 2 dimensions.
Example:
3D array:
int box[3][4][2];
Logic
Represents data as:

- 3 layers
- Each layer has 4 rows
- Each row has 2 elements

Used in:

- Scientific simulations
- 3D graphics
- Multi-level data tables

Example access:
box[i][j][k]