UNIVERSITY OF MILA

Faculty of Science and Technology

Department of Process Engineering

Level: 1st year ST - ENG & LMD

Course : Introduction to Programming

Lectures 07:

Multi-Dimensional Arrays and Matrix Operations

---

by:

Dr. Farouk KECITA

Academic Year 2025/2026

1 Multi-Dimensional Arrays

Array having more than one subscript variable is called Multi-Dimensional array; it stores homogeneous data in tabular form. In general, data in multidimensional arrays is stored in row-major order in memory.

1.1 Declaration of N-Dimensional Arrays

The general form of declaring N-dimensional arrays is:

Syntax: `<data type> <array name> [size1][size2]...[sizeN];`

Examples:

- Two dimensional array: `int arr[5][10];`
- Three dimensional array: `int arr[5][10][20];`

1.2 Size of Multi-dimensional Arrays

The total size of a multi-dimensional array, which is the total number of elements that can be stored, can be calculated by multiplying the size of all its dimensions.

Examples:

- Two dimensional array: `int arr[5][10]` can store total $(5 \times 10) = 50$ elements.
- Three dimensional array: `int arr[5][10][10]` can store total $(5 \times 10 \times 10) = 500$ elements.

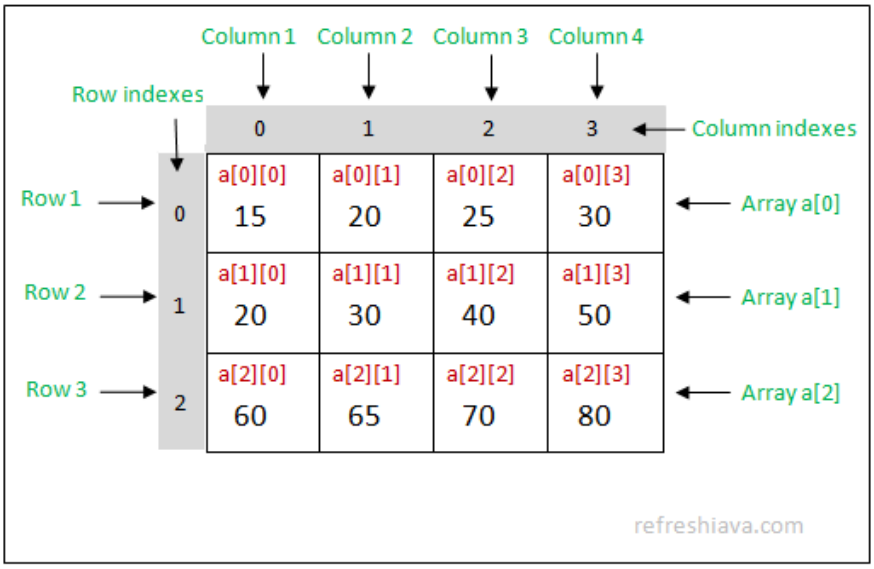
In order to get the size of the multi-dimensional in bytes, we multiply the size of a single element with the total number of elements.

- Size of 2D array: `int arr[5][10]` $\Rightarrow 5 \times 10 \times 2 = 100$ bytes.
- Size of 3D array: `int arr[5][10][10]` $\Rightarrow 5 \times 10 \times 10 \times 2 = 1000$ bytes (assuming `int` $\Leftrightarrow$ 2 bytes).

The most commonly used forms of the multi-dimensional array are: Two Dimensional and Three Dimensional Arrays.

2 Two-Dimensional Array (Matrix)

A two-dimensional array (2D array or matrix) in the C language is the simplest form of a multidimensional array that requires two subscript variables; one subscript variable denotes the "Row" of the matrix, and another subscript variable denotes its "Column". We can visualize a two-dimensional array as an array of one-dimensional arrays arranged one over another, forming a table with 'size1' rows and 'size2' columns where the row number ranges from 0 to (size1-1) and the column number ranges from 0 to (size2-1).



2.1 Declaration of Two-Dimensional Array in C

The basic form of declaring a 2D array with size1 rows and size2 columns in C is shown below.

Syntax: `<data type> arrayName[size1][size2];`

- `data_type` can be int, float, char, double.
- `arrayName` is the name of the array.
- `size1` and `size2` indicates number of rows and columns respectively.

Example: We can declare a two-dimensional integer array say 'arr' with 4 rows and 5 columns as: `int arr[4][5];`
The above statements allocates memory for $4 \times 5 = 20$ elements i.e $20 \times 2 = 40$ bytes.

In this kind of declaration, the 2D array is allocated memory in the stack and its size should be known at the compile time i.e. size of the 2D array is fixed.

2.2 Memory Representation

2D arrays are stored in contiguous memory location row-wise. A 3×3 Array is shown in the above diagram.

Consider a 3×3 Array stored in contiguous memory location starting from address 5000:

- Array element `arr[0][0]` will be stored at address 5000
- `arr[0][1]` will be stored at the next memory location
- After elements of first row are stored, elements of the next row get their corresponding memory locations

Memory Address Calculation:

$$\text{Address of } arr[0][i] = \text{Address of } arr[0][i-1] + \text{Size of Data Type}$$

Example: Consider a two-dimensional array of integers consisting of 1 row and 3 columns: `arr[1][3];`

Array element	Memory location
arr[0][0]	5000
arr[0][1]	5002
arr[0][2]	5004

2.3 Initialization of Two-Dimensional Arrays in C

There are several methods to initialize a 2D array.

2.3.1 Initializing All Elements Row-Wise

In this type of initialization, we use nested braces. A row is represented by each set of inner braces.

Syntax: `int arr[3][3] = {{1,2,3},{4,5,6},{7,8,9}};`

```c
#include <stdio.h>
int main() {
    int i, j;
    int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    printf("array elements are:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }   return 0;   }
```

Listing 1: Row-Wise Initialization

```
array elements are:
1 2 3
4 5 6
7 8 9
```

2.3.2 Combined Initialization

We can initialize a 2D array using an initializer list, where all values are set sequentially.

Syntax: `int arr[3][3] = {1,2,3,4,5,6,7,8,9};`

```c
#include <stdio.h>
int main() {
    int i, j;
    int arr[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

    printf("array elements are:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }    return 0;  }
```

Listing 2: Combined Initialization

2.3.3 Partial 2D Array Initialization

In this case, not all elements are initialized. Uninitialized elements will have the default value of 0.

Syntax: `int arr[3][3] = {{1},{5,6},{9}};`

```c
#include <stdio.h>
int main() {
    int i, j;
    int arr[3][3] = {{1}, {5, 6}, {9}};

    printf("array elements are:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Listing 3: Partial Initialization

```
array elements are:
1 0 0
5 6 0
9 0 0
```

2.4 Accessing Elements of 2D Array in C

To access elements in 2D arrays, you need to specify the subscript variables (indices) for each dimension: row index and column index.

Syntax: `Array_Name[i][j];`
Example: `arr[2][1];`

2.5 Accepting and Printing a 2D Array in C

For storing and printing the whole two-dimensional array, we access each element using two nested loops. The outer loop represents rows, while the inner loop represents columns.

```c
#include <stdio.h>
int main() {
    int i, j, NR = 2, NC = 2;
    int arr[NR][NC];
    printf("Enter the elements of array\n");
    for (i = 0; i < NR; i++) {
        for (j = 0; j < NC; j++) {
            scanf("%d", &arr[i][j]);
        }
    }
    printf("The array elements are: \n");
    for (i = 0; i < NR; i++) {
        for (j = 0; j < NC; j++) {
            printf("arr[%d][%d] = %d ", i, j, arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Listing 4: Accept and Print a 2D Array

```
Enter the elements of array
1 2 3 4
The array elements are:
arr[0][0] = 1 arr[0][1] = 2
arr[1][0] = 3 arr[1][1] = 4
```

3 Matrix Operations in C

3.1 Copying a Matrix to Another

Copying an array involves iterating over each row and column using two nested loops and copying each element to its correct position in the destination array. The destination array must be at least the same size as the original array.

```c
#include <stdio.h>
#define MAXROW 100
#define MAXCOL 100

int main() {
    int m1[MAXROW][MAXCOL], m2[MAXROW][MAXCOL];
    int i, j, n_row, n_col;

    // Get the number of rows and columns from the user
    printf("Enter the number of rows (0-100): ");
    scanf("%d", &n_row);
    printf("Enter the number of columns (0-100): ");
    scanf("%d", &n_col);

    // Boundary check
    if (n_row > MAXROW || n_col > MAXCOL || n_row < 0 || n_col
       < 0) {
        printf("Warning !! Boundary Level Exceeded.\n");
        return 0;
    }

    // Get the entries for the original matrix
    printf("Enter the elements of the original matrix:\n");
    for (i = 0; i < n_row; i++) {
        for (j = 0; j < n_col; j++) {
            printf("m1[%d][%d] = ", i, j);
            scanf("%d", &m1[i][j]);
        }
    }

    // Copy matrix m1 to m2
    for (i = 0; i < n_row; i++) {
        for (j = 0; j < n_col; j++) {
            m2[i][j] = m1[i][j];
        }
    }

    // Display the original matrix
    printf("\nOriginal matrix (m1):\n");
    for (i = 0; i < n_row; i++) {
        for (j = 0; j < n_col; j++) {
            printf("%d\t", m1[i][j]);
```

```c
        }
        printf("\n");
    }

    // Display the copied matrix
    printf("\nCopied matrix (m2):\n");
    for (i = 0; i < n_row; i++) {
        for (j = 0; j < n_col; j++) {
            printf("%d\t", m2[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Listing 5: Program to Copy One Matrix to Another

```
Enter the number of rows (0-100): 2
Enter the number of columns (0-100): 2
Enter the elements of the original matrix:
m1[0][0] = 1
m1[0][1] = 2
m1[1][0] = 3
m1[1][1] = 4

Original matrix (m1):
1 2
3 4

Copied matrix (m2):
1 2
3 4
```

3.2 Changing an Element in a Matrix

To change the value of a specific element in a matrix, we refer to its indices (row and column) and assign a new value.

3.2.1 Static Method

```c
#include <stdio.h>
int main() {
    int matrix[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

```

```c
    printf("Original matrix:\n");
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Change element at row 1, column 0 (second row, first
       column)
    matrix[1][0] = 10;

    printf("\nMatrix after change:\n");
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Listing 6: Static Method to Change an Element

```
Original matrix:
1 2 3
4 5 6
7 8 9

Matrix after change:
1 2 3
10 5 6
7 8 9
```

3.2.2 Dynamic Method (User Input)

```c
#include <stdio.h>
int main() {
    int mat[3][3];
    int i, j, target, row, col;

    printf("Enter elements into mat (3x3):\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            scanf("%d", &mat[i][j]);
        }
    }

    printf("Enter the row and column number of target element:
       ");
```

```c
    scanf("%d %d", &row, &col);

    printf("Enter the new value: ");
    scanf("%d", &target);

    // Change the element
    mat[row][col] = target;

    printf("Elements of the output matrix are:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Listing 7: Dynamic Method to Change an Element

```
Enter elements into mat (3x3):
1 2 3 4 5 6 7 8 9
Enter the row and column number of target element: 1 2
Enter the new value: 10
Elements of the output matrix are:
1 2 3
4 5 10
7 8 9
```

3.3 Arithmetic Operations on Matrices

Basic mathematical operations such as addition, subtraction, multiplication, and division can be performed on matrices element-by-element.

General Syntax: If A and B are two $(m \times n)$ matrices:

- Addition: `C[i][j] = A[i][j] + B[i][j]`
- Subtraction: `C[i][j] = A[i][j] - B[i][j]`
- Element-wise Multiplication: `C[i][j] = A[i][j] * B[i][j]`
- Element-wise Division: `C[i][j] = A[i][j] / B[i][j]`

3.3.1 Matrix Addition

We can add matrices together as long as they have the same number of rows and columns. To add matrices, we add the corresponding elements from both matrices.

$$(A+B)_{ij} = A_{ij} + B_{ij} \quad \text{where } 1 \leq i \leq m \text{ and } 1 \leq j \leq n$$

Example:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

```c
#include <stdio.h>
int main() {
    int mat1[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int mat2[3][3] = {
        {9, 10, 11},
        {12, 13, 14},
        {15, 16, 17}
    };
    int sum[3][3], i, j;
    printf("The first matrix is:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d\t", mat1[i][j]);
        }
        printf("\n");
    }
    printf("\nThe second matrix is:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d\t", mat2[i][j]);
        }
        printf("\n");
    }
    // Add the two matrices
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            sum[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
    printf("\nThe sum of the two matrices is:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d\t", sum[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Listing 8: Program to Add Two Matrices

```
The first matrix is:
1 2 3
4 5 6
7 8 9

The second matrix is:
9 10 11
12 13 14
15 16 17

The sum of the two matrices is:
10 12 14
16 18 20
22 24 26
```

3.3.2 Matrix Multiplication

Condition for Matrix Multiplication: To multiply matrix A by matrix B, the number of columns in A must equal the number of rows in B.

If A is of size $m \times n$ and B is of size $n \times p$, then the product C = A × B will be of size $m \times p$.

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$

Example:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix} = \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

$$(1 \times 5 + 2 \times 8 = 21, \quad 1 \times 6 + 2 \times 9 = 24, \quad 1 \times 7 + 2 \times 10 = 27)$$

$$(3 \times 5 + 4 \times 8 = 47, \quad 3 \times 6 + 4 \times 9 = 54, \quad 3 \times 7 + 4 \times 10 = 61)$$

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int A[10][10], B[10][10], M[10][10];
    int m1, n1, m2, n2, i, j, k;

    printf("Enter the number of rows and columns of matrix A:
        ");
    scanf("%d%d", &m1, &n1);

    printf("Enter the number of rows and columns of matrix B:
        ");
    scanf("%d%d", &m2, &n2);

```

```c
    // Check if multiplication is possible
    if(n1 != m2) {
        printf("Warning !! Matrix multiplication is not
           possible.\n");
        printf("Number of columns in A must equal number of
           rows in B.\n");
        return 0;
    }

    // Input matrix A
    printf("Enter elements of matrix A:\n");
    for(i = 0; i < m1; i++) {
        for(j = 0; j < n1; j++) {
            scanf("%d", &A[i][j]);
        }
    }

    // Input matrix B
    printf("Enter elements of matrix B:\n");
    for(i = 0; i < m2; i++) {
        for(j = 0; j < n2; j++) {
            scanf("%d", &B[i][j]);
        }
    }

    // Matrix multiplication
    for(i = 0; i < m1; i++) {
        for(j = 0; j < n2; j++) {
            M[i][j] = 0;
            for(k = 0; k < n1; k++) {
                M[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    // Display result
    printf("\nProduct of the two matrices is:\n");
    for(i = 0; i < m1; i++) {
        for(j = 0; j < n2; j++) {
            printf("%d\t", M[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Listing 9: Program for Matrix Multiplication

```
Enter the number of rows and columns of matrix A: 2 2
Enter the number of rows and columns of matrix B: 2 3
Enter elements of matrix A:
1 2 3 4
Enter elements of matrix B:
5 6 7 8 9 10

Product of the two matrices is:
21 24 27
47 54 61
```

3.4 Matrix Addition with User Input

```c
#include <stdio.h>
int main() {
    int A[10][10], B[10][10], sum[10][10];
    int rows, cols, i, j;
    printf("Enter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);

    printf("Enter elements of first matrix:\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++) {
            scanf("%d", &A[i][j]);
        }
    }
    printf("Enter elements of second matrix:\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++) {
            scanf("%d", &B[i][j]);
        }
    }
    // Add matrices
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++) {
            sum[i][j] = A[i][j] + B[i][j];
        }
    }
    printf("Sum of matrices:\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++) {
            printf("%d ", sum[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Listing 10: Matrix Addition with Dynamic Sizes

Summary of Matrix Operations

Operation	Description	Condition
Copy	Duplicate matrix	Same size
Modify	Change element value	Valid indices
Addition	Add corresponding elements	Same dimensions
Multiplication	Multiply rows by columns	cols(A) = rows(B)

Table 1: Summary of Matrix Operations

Key Takeaways:

- Matrix operations require careful index management
- Always check matrix dimensions before operations
- Matrix multiplication requires cols(A) = rows(B)
- Nested loops are essential for matrix manipulation