Lecture 07: Multi-Dimensional Arrays and Matrix Operations in C

Course: Introduction to Programming

Dr. Farouk KECITA

Department of Process Engineering
Faculty of Science and Technology
University of Mila
Level: 1st year ST - ENG & LMD

Academic Year 2025/2026

Outline

Multi-Dimensional Arrays

Definition

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

Multi-Dimensional Arrays

Definition

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

Examples

- `int arr[5][10];` (2D array)
- `int arr[5][10][20];` (3D array)

Multi-Dimensional Arrays

Definition

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

Examples

- `int arr[5][10];` (2D array)
- `int arr[5][10][20];` (3D array)

Syntax

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

Size of Multi-dimensional Arrays

Total Elements

Total elements = size1 x size2 x ... x sizeN

2D Array

`int arr[5][10]`

Elements: $5 \times 10 = 50$

Size in bytes: $50 \times 2 = 100$ bytes

Size of Multi-dimensional Arrays

Total Elements

Total elements = size1 x size2 x ... x sizeN

2D Array

`int arr[5][10]`

Elements: $5 \times 10 = 50$

Size in bytes: $50 \times 2 = 100$ bytes

3D Array

`int arr[5][10][10]`

Elements: $5 \times 10 \times 10 = 500$

Size in bytes: $500 \times 2 = 1000$ bytes

Size of Multi-dimensional Arrays

Total Elements

Total elements = size1 x size2 x ... x sizeN

2D Array

`int arr[5][10]`

Elements: $5 \times 10 = 50$

Size in bytes: $50 \times 2 = 100$ bytes

3D Array

`int arr[5][10][10]`

Elements: $5 \times 10 \times 10 = 500$

Size in bytes: $500 \times 2 = 1000$ bytes

Note

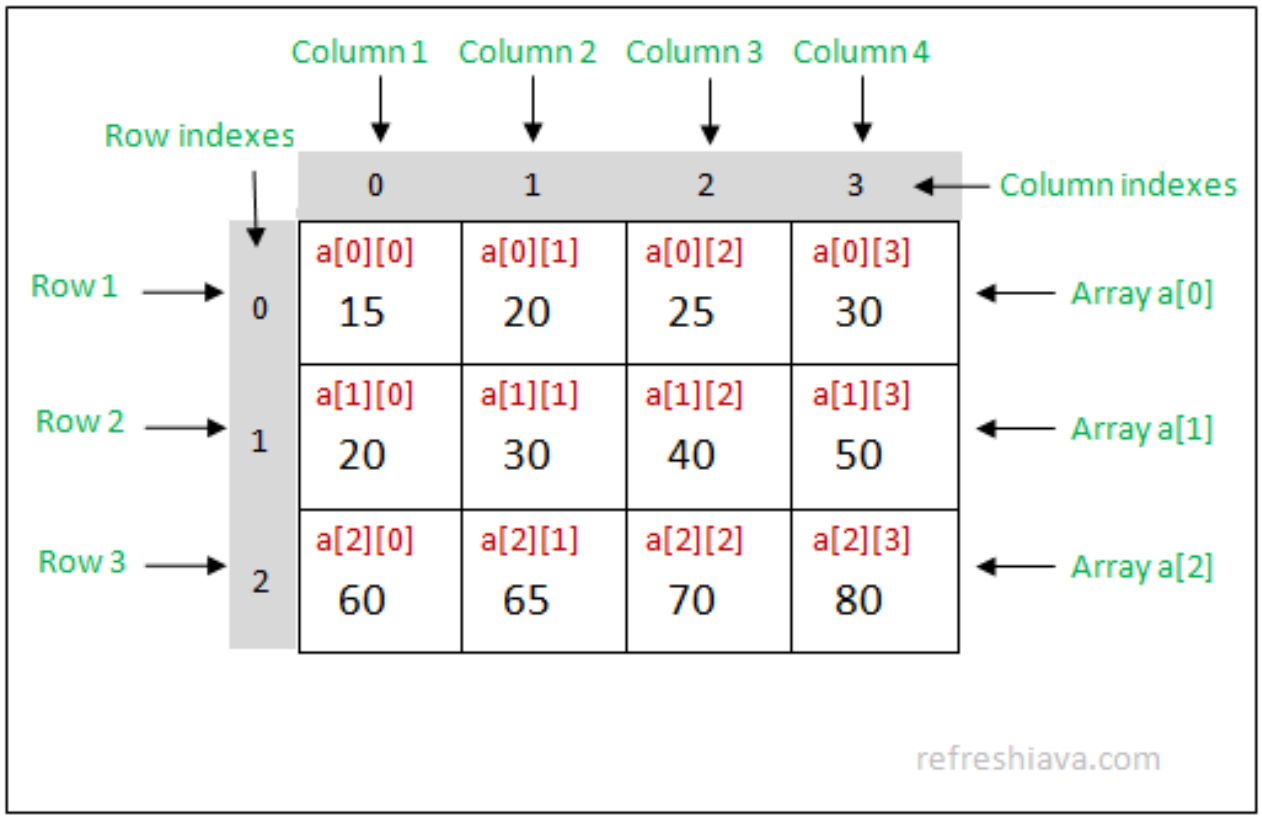
Assuming `int` occupies 2 bytes in memory.

Two-Dimensional Array (Matrix)

Definition

A 2D array requires two subscript variables:

- First index: **Row** (0 to size1-1)
- Second index: **Column** (0 to size2-1)



Declaration of 2D Arrays

Syntax

```
<data type> arrayName[rows][columns];
```

Example

```
int matrix[4][5];
```

- 4 rows (indices 0-3)
- 5 columns (indices 0-4)
- Total elements: $4 \times 5 = 20$
- Memory: $20 \times 2 = 40$ bytes

Note

Size must be known at compile time (stack allocation).

Memory Representation

Row-Major Order

2D arrays are stored in contiguous memory locations row by row.

Example: 3×3 Array starting at address 5000

Element	**Address**
arr[0][0]	5000
arr[0][1]	5002
arr[0][2]	5004
arr[1][0]	5006
arr[1][1]	5008
arr[1][2]	5010

Address Calculation

Address of arr[0][i] = Address of arr[0][i-1] + Size of Data Type

Initialization Methods for 2D Arrays

Row-Wise

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

Initialization Methods for 2D Arrays

Row-Wise

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

Combined

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

Initialization Methods for 2D Arrays

Row-Wise

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

Partial

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

Combined

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

Initialization Methods for 2D Arrays

Row-Wise

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

Partial

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

Combined

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

Partial Initialization Result

$$\begin{pmatrix} 1 & 0 & 0 \\ 5 & 6 & 0 \\ 9 & 0 & 0 \end{pmatrix}$$

Row-Wise Initialization - Example

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

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

Output

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

Partial Initialization - Example

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

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

Output

```
Array elements:
1 0 0
5 6 0
9 0 0
```

Accessing and Printing 2D Arrays

```c
#include <stdio.h>
int main() {
    int arr[2][2];

    printf("Enter 4 elements:\n");
    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 2; j++) {
            scanf("%d", &arr[i][j]);
        }
    }

    printf("Matrix:\n");
    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 2; j++) {
            printf("arr[%d][%d]=%d ", i, j, arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Copying a Matrix

Process

Copy each element from source to destination using nested loops.

Important

Direct assignment `matrix2 = matrix1;` is NOT allowed!

Syntax

```
for(i = 0; i < rows; i++)
    for(j = 0; j < cols; j++)
        dest[i][j] = source[i][j];
```

Matrix Copy - Complete Program

```
1  # include <stdio.h>
2  # define MAXROW 100
3  # define MAXCOL 100
4
5  int main () {
6   int m1[ MAXROW ][ MAXCOL ], m2[ MAXROW ][ MAXCOL ],i, j,
        n_row , n_col ;
7
8  printf (" Enter rows and columns (0 -100) : ");
9  scanf ("%d%d", &n_row , & n_col );
10
11  if( n_row > MAXROW || n_col > MAXCOL ) {
12  printf (" Boundary exceeded !\n");
13  return 0; }
14
15  printf (" Enter elements :\n");
16  for(i = 0; i < n_row; i++)  {
17          for(j = 0; j < n_col; j++){
18  scanf ("%d", &m1[i][j]);
19           }
20      }
```

Matrix Copy - Complete Program

```
// Copy matrix

     for(i = 0; i < n_row; i++)  {
          for(j = 0; j < n_col; j++){
               m2[i][j] = m1[i][j];
           }}
     printf("\nOriginal matrix:\n");
     for(i = 0; i < n_row; i++) {
          for(j = 0; j < n_col; j++){
               printf("%d\t", m1[i][j]);
             }
          printf("\n");
     }

     printf("\nCopied matrix:\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;
      }
```

Modifying Matrix Elements

Static Modification

```
1 int matrix[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
2 matrix[1][0] = 10;  // Change element at row 1, col 0
```

Result

$$\begin{pmatrix} 1 & 2 & 3 \\ 10 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Dynamic Element Modification

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

    printf("Enter 9 elements:\n");
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            scanf("%d", &mat[i][j]);
        }   }
    printf("Enter row and column to change: ");
    scanf("%d%d", &row, &col);
    printf("Enter new value: ");
    scanf("%d", &target);

    mat[row][col] = target;

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

Matrix Addition

Formula

$$(A + B)_{ij} = A_{ij} + B_{ij}$$

Example

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

Condition

Both matrices must have the same dimensions.

Matrix Addition - Program

```c
#include <stdio.h>

int main() {
    int A[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int B[3][3] = {{9,10,11},{12,13,14},{15,16,17}};
    int sum[3][3];

    // Add matrices
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            sum[i][j] = A[i][j] + B[i][j];
    printf("Sum of matrices:\n");
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++){
            printf("%d\t", sum[i][j]);
            }
        printf("\n");
    }
    return 0;
}
```

Output

```
10   12   14
16   18   20
```

Matrix Multiplication

Formula

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

Important Condition

Number of columns in A must equal number of rows in B!

Example

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

Matrix Multiplication - Algorithm

Steps

1. Check if cols(A) == rows(B)
2. Initialize result matrix C of size rows(A) X cols(B)
3. For i = 0 to rows(A)-1
4. For j = 0 to cols(B)-1
5. C[i][j] = 0
6. For k = 0 to cols(A)-1
7. C[i][j] += A[i][k] * B[k][j]

Matrix Multiplication - Program

```c
#include <stdio.h>

int main() {

    int A[10][10], B[10][10], C[10][10];
    int m1, n1, m2, n2;

    printf("Enter dimensions of A: ");
    scanf("%d%d", &m1, &n1);

    printf("Enter dimensions of B: ");
    scanf("%d%d", &m2, &n2);

    if(n1 != m2) {
        printf("Multiplication not possible!\n");
        return 0;
         }

    printf("Enter A elements:\n");

    for(int i = 0; i < m1; i++){
        for(int j = 0; j < n1; j++){
            scanf("%d", &A[i][j]);
   }  }
```

Matrix Multiplication - Program

```
1
2       printf("Enter B elements:\n");
3
4       for(int i = 0; i < m2; i++)
5           for(int j = 0; j < n2; j++){
6               scanf("%d", &B[i][j]);{
7            }}
8       // Multiply matrices
9
10      for(int i = 0; i < m1; i++) {
11          for(int j = 0; j < n2; j++) {
12              C[i][j] = 0;
13              for(int k = 0; k < n1; k++){
14                  C[i][j] += A[i][k] * B[k][j];
15                  }
16          }
17      }
18      printf("Product:\n");
19      for(int i = 0; i < m1; i++) {
20          for(int j = 0; j < n2; j++){
21                      printf("%d\t", C[i][j]);
22                       }
23          printf("\n");
24       }
25      return 0;
26       }
```

Matrix Addition - Dynamic Sizes

```c
#include <stdio.h>
int main() {
    int A[10][10], B[10][10], sum[10][10];
    int rows, cols;
        printf("Enter rows and columns: ");
    scanf("%d%d", &rows, &cols);
       printf("Enter first matrix:\n");
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++){
            scanf("%d", &A[i][j]);
             }  }
    printf("Enter second matrix:\n");
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            scanf("%d", &B[i][j]);
            }  }
    // Add matrices
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            sum[i][j] = A[i][j] + B[i][j];
             }  }
    printf("Sum:\n");
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++){
            printf("%d ", sum[i][j]);
             }
        printf("\n");
    }
    return 0;  }
```

Summary of Matrix Operations

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

Summary of Matrix Operations

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

Key Takeaways

- Always check matrix dimensions before operations
- Nested loops are essential for matrix manipulation
- 2D arrays are stored in row-major order
- Array indices start at 0 in C