Multidimensional Arrays

1. Basics of Multidimensional Arrays

- **One-Dimensional Array (1D)**: A simple list of elements, like a row of numbers.
- **Two-Dimensional Array (2D)**: Often visualized as a table or matrix with rows and columns. Each element is accessed by specifying both a row index and a column index.
- **Three-Dimensional Array (3D)**: Extends the concept of 2D arrays to include depth, like a stack of matrices.
- **Higher-Dimensional Arrays**: Arrays can have more than three dimensions, though they are harder to visualize. These are used for more complex data structures.

2. **Declaration and Initialization**

C/C++:

```
int array[3][4]; // 2D array with 3 rows and 4 columns

int array[2][3][4]; // 3D array with 2 layers, 3 rows, and 4 columns
```

Java:

```
int[][] array = new int[3][4]; // 2D array

int[][][] array = new int[2][3][4]; // 3D array
```

Python:

```
import numpy as np

array = np.zeros((3, 4)) # 2D array with 3 rows and 4 columns

array = np.zeros((2, 3, 4)) # 3D array with 2 layers, 3 rows, and 4 columns
```

3. Accessing Elements

- Elements in a multidimensional array are accessed using multiple indices:
  - **2D Array**: array[row][col]
  - **3D Array**: array[depth][row][col]

Applications

- **Matrices**: Representing and performing operations on matrices in mathematics.
- **Grids**: For games, simulations, or any application where data is laid out in a grid.
- **Tensor Representation**: In machine learning and data science, multidimensional arrays (often called tensors) are used to store and manipulate data.
- **Image Processing**: Images are represented as 2D or 3D arrays (for colored images).

5. Example: Matrix Multiplication

- Given two 2D arrays (matrices) A and B, their product C is a new matrix where each element is computed by taking the dot product of the corresponding row in A and column in B.

Pseudocode:

```
for i from 1 to n:
  for j from 1 to m:
    C[i][j] = 0
    for k from 1 to p:
      C[i][j] += A[i][k] * B[k][j]
```

6. Memory Layout

- **Row-Major Order**: Common in languages like C and C++, where consecutive elements of a row are stored next to each other in memory.
- **Column-Major Order**: Used by some languages like Fortran, where consecutive elements of a column are stored next to each other.

2D Array of Integers:

Java:

```
public class Main {
  public static void main(String[] args) {
    int[][] matrix = new int[3][3];
```

```
        // Filling the matrix
        int value = 1;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                matrix[i][j] = value++;
            }
        }
        // Printing the matrix
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

C++ :

```
#include <iostream>
using namespace std;

int main() {
    int matrix[3][3];

    // Filling the matrix
    int value = 1;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            matrix[i][j] = value++;
        }
    }

    // Printing the matrix
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Python:

```
# Creating a 3x3 matrix
matrix = [[0 for _ in range(3)] for _ in range(3)]

# Filling the matrix
value = 1
for i in range(3):
    for j in range(3):
        matrix[i][j] = value
        value += 1

# Printing the matrix
for row in matrix:
    print(" ".join(map(str, row)))
```