



جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الأمن
السيبراني
DEPARTMENT OF CYBER SECURITY

SUBJECT:

STRUCTURED PROGRAMMING

CLASS:

1ST STAGE

LECTURER:

DR. ABDULKADHEM A. ABDULKADHEM

LECTURE: (7)

Two-Dimensional Arrays



1. Introduction to Two-Dimensional Arrays

A **two-dimensional array** (2D array) can be thought of as a table, where data is arranged in rows and columns. It is essentially an array of arrays.

In C++, a two-dimensional array is declared by specifying two sizes: one for the rows and one for the columns.

2. Declaring a Two-Dimensional Array

The syntax to declare a 2D array is as follows:

General Form of 2D-Array:

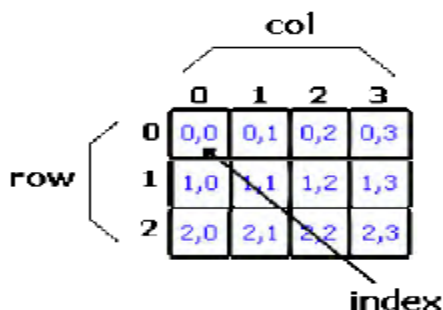
```
data-type Array-name [ Row-size ] [ Col-size ];
```

Where:

- **datatype:** Type of elements in the array (e.g., int, float, char).
- **Array_name:** Name of the array.
- **row-size:** Number of rows in the 2D array.
- **col-size:** Number of columns in the 2D array.

Example:

```
int    a [10] [10];  
int    num [3] [4];
```



Here, **num** is a 2D array with 3 rows and 4 columns.



3. Initializing a Two-Dimensional Array

You can initialize a 2D array at the time of declaration by specifying the values for all elements in the array.

a. Static Initialization

Example:

```
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

This initializes the array `matrix` with two rows and three columns.

- Row 1: {1, 2, 3}
- Row 2: {4, 5, 6}

1	2	3
4	5	6

b. Partial Initialization

If the number of elements specified is less than the total size of the array, the remaining elements are automatically initialized to zero.

Example:

```
int matrix[3][3] = {{1, 2}, {3, 4}};
```

This will initialize:

- Row 1: {1, 2, 0}
- Row 2: {3, 4, 0}
- Row 3: {0, 0, 0}

c. Zero Initialization

You can initialize the entire array to zero by specifying `{0}`:

```
int matrix[3][3] = {0}; // All elements set to 0
```

4. Accessing Elements of a Two-Dimensional Array

You access elements of a 2D array using two indices: one for the row and one for the column.

Syntax:

```
array_name[row_index][column_index]
```



Example:

```
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};  
cout << matrix[1][2]; // Accessing element at row 1, column 2 (output: 6)
```

Here, `matrix[1][2]` refers to the element at row 1 (second row) and column 2 (third column), which is 6.

5. Reading and Printing Two-Dimensional Array Elements

You can read and print elements of a 2D array using nested loops. The outer loop will iterate over the rows, and the inner loop will iterate over the columns.

Example:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};  
  
    // Reading and printing the 2D array elements  
    for (int i = 0; i < 2; i++) {  
        for (int j = 0; j < 3; j++) {  
            cout << "Element at [" << i << "][" << j << "] is " << matrix[i][j]  
            << endl;  
        }  
    }  
  
    return 0;  
}
```

Output:

```
Element at [0][0] is 1  
Element at [0][1] is 2  
Element at [0][2] is 3  
Element at [1][0] is 4  
Element at [1][1] is 5  
Element at [1][2] is 6
```

6. Modifying Elements of a Two-Dimensional Array

You can modify the elements of a 2D array just like with one-dimensional arrays, by using the appropriate row and column indices.

Example:



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

int main() {
    int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

    // Modify element at row 1, column 2
    matrix[1][2] = 100;

    // Print the updated matrix
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output:

```
1 2 3
4 5 100
```

7. Processing Elements of a Two-Dimensional Array

You can perform various operations on the elements of a 2D array, such as summing all elements, finding the maximum element, etc.

Example: Calculating the sum of all elements in a 2D array.

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

int main() {
    int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
    int sum = 0;

    // Calculate sum of all elements
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            sum += matrix[i][j];
        }
    }

    cout << "The sum of all elements in the matrix is: " << sum << endl;

    return 0;
}
```



Output:

```
The sum of all elements in the matrix is: 21
```

8. Important Notes on Two-Dimensional Arrays

- **Memory Layout:** A 2D array in C++ is stored in row-major order, meaning the elements of each row are stored contiguously in memory.
- **Bounds Checking:** C++ does not perform bounds checking. Accessing an element outside the array's declared size will result in undefined behavior.

Conclusion

In this lecture, we covered:

- How to declare and initialize two-dimensional arrays.
- How to access, modify, read, and print elements in a 2D array.
- Basic operations like summing array elements.

Two-dimensional arrays are used when dealing with problems that involve matrices or grids, such as image processing, board games, and data representation in tables.