Lec 8

Multi-Dimensional Arrays

Multi-dimensional arrays have two or more index values which are used to specify a particular element in the array.

For this 2D array element,image[i][j]

the first index value i specifies a row index,while j specifies a column index.

Declaring multi-dimensional arrays is similar to the 1D case:

int a[10]; declare 1D

float b[3][5]; declare 2D array

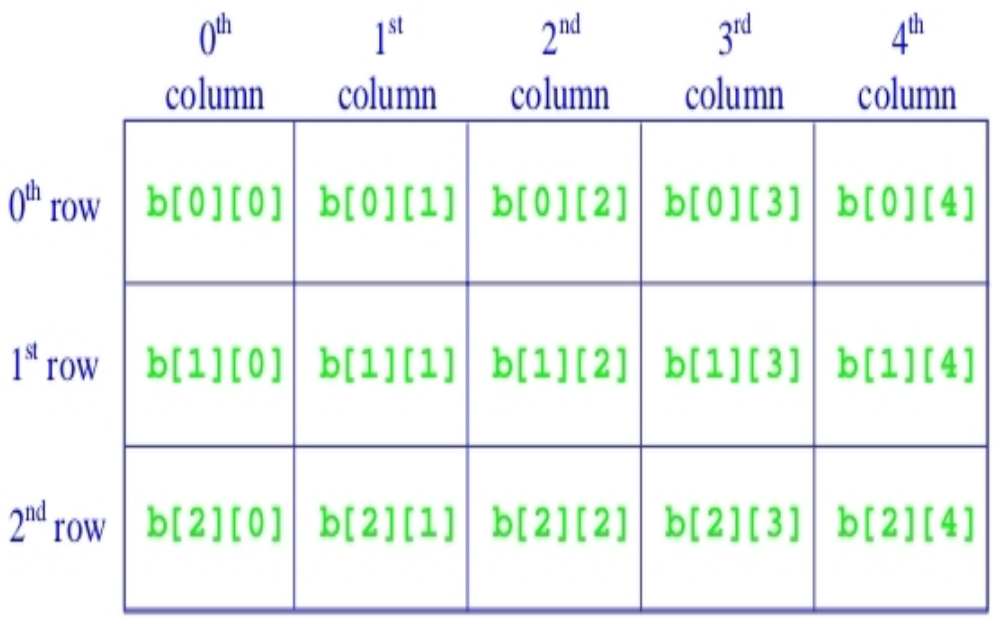
double c[6][4][2]; declare 3D array

Note that it is quite easy to allocate a large chunk of consecutive memory with multi-dimensional arrays.

Array c contains 6×4×2=48 doubles.

Multi-Dimensional Array Illustration

A useful way to picture a 2D array is as a grid or matrix. Picture array b as



In C, 2D arrays are stored by row.

Which means that in memory the 0th row is put into its memory locations, the 1st row then takes up the next memory locations, the 2nd row takes up the next memory location and so on...

Initializing Multi-Dimensional Arrays

This procedure is entirely analogous to that used to initialize 1D arrays at their declaration.

For example, this declaration static

int age[2][3]={4,8,12,19,6,-1};

will fill up the array age as it is stored in memory.

That is, the array is initialized row by row. Thus, the above statement is equivalent to:

age[0][0]=4; age[0][1]=8; age[0][2]=12;

age[1][0]=19; age[1][1]=6; age[1][2]=-1;

As before, if there are fewer initialization values than array elements, the remainder are initialized to zero.

To make your program more readable, you can explicitly put the values to be

assigned to the same row in inner curly brackets:

```
static int age[2][3]={{4,8,12},{19,6,-1}};
```

In addition if the number of rows is omitted from the actual declaration, it is set equal to the number of inner brace pairs:

```
static int age[][3]={{4,8,12},{19,6,-1}};
```

Basic 2D Array: Printing a Matrix

This code is the simplest example of defining a two-dimensional array and printing it in a table format.

```
#include <stdio.h>

int main() {

    // Declare and initialize a 2D array with 3 rows and 3 columns

    int matrix[3][3] = {

,{3 ,2 ,1}

,{6 ,5 ,4}

{9 ,8 ,7}

    };

    printf("The Matrix elements are:\n");

    // Outer loop for rows

    for(int i = 0; i < 3; i++) {

        // Inner loop for columns

        for(int j = 0; j < 3; j++) {
```

```
            printf("%d\t", matrix[i][j]);
        }
        printf("\n"); // Move to next line after
each row
    }
    return 0;
}
```

User Input: Summing All Elements

In this example, we ask the user to enter values, and then we calculate the sum of all numbers within the Matrix.

```
#include <stdio.h>

int main() {
    int rows = 2, cols = 2;
    int arr[2][2];
    int sum = 0;
    printf("Enter 4 numbers for a 2x2 matrix:\n");
    // Taking input from user
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++) {
            printf("Enter element at [%d][%d]: ", i, j);
            scanf("%d", &arr[i][j]);
        }
```

```
    }

    // Calculating sum
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++) {
            sum += arr[i][j];
        }
    }

    printf("\nThe sum of all elements is:
 %d\n", sum);
    return 0;
}
```

Matrix Addition

This code demonstrates how to add two matrices and store the result in a third matrix (a mathematical operation).

```
#include <stdio.h>

int main() {
    int A[2][2] = {{5, 10}, {15, 20}};
    int B[2][2] = {{1, 2}, {3, 4}};
    int result[2][2];

    // Adding elements of A and B
    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 2; j++) {
            result[i][j] = A[i][j] + B[i][j];
        }
    }
```

```
    printf("Result of Matrix Addition (A + B):\n");
    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 2; j++) {
            printf("%d  ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Key Concepts:

Declaration: The syntax is type name[rows][columns];.

Indexing: Indexing always starts from 0.

The first element is [0][0].

Nested Loops: Nested loops are always required to handle two-dimensional arrays; the outer loop manages the rows, and the inner loop manages the columns.

Homework #1: Find the Maximum Value

Objective: Write a program to find the largest number in a 3 x 3 matrix.

Problem Description:

Define a 3 x 3 integer array with values of your choice.

Create a variable called max and initialize it with the first element of the array.

Use nested loops to compare every element with max.

Print the maximum value found.

Homework #2: Row-wise Summation

Objective: Calculate the sum of each row individually.

Problem Description:

Define a 2 x 4 array (2 rows, 4 columns).

For each row, calculate the total sum of its elements.

Print the result for each row separately (e.g., "Sum of row 0: 25").

Homework #3: Identity Matrix Checker

Objective: Check if a 3 x 3 matrix entered by the user is an Identity Matrix.

Mathematical Rule: In an Identity Matrix, all diagonal elements are 1, and all other elements are 0.