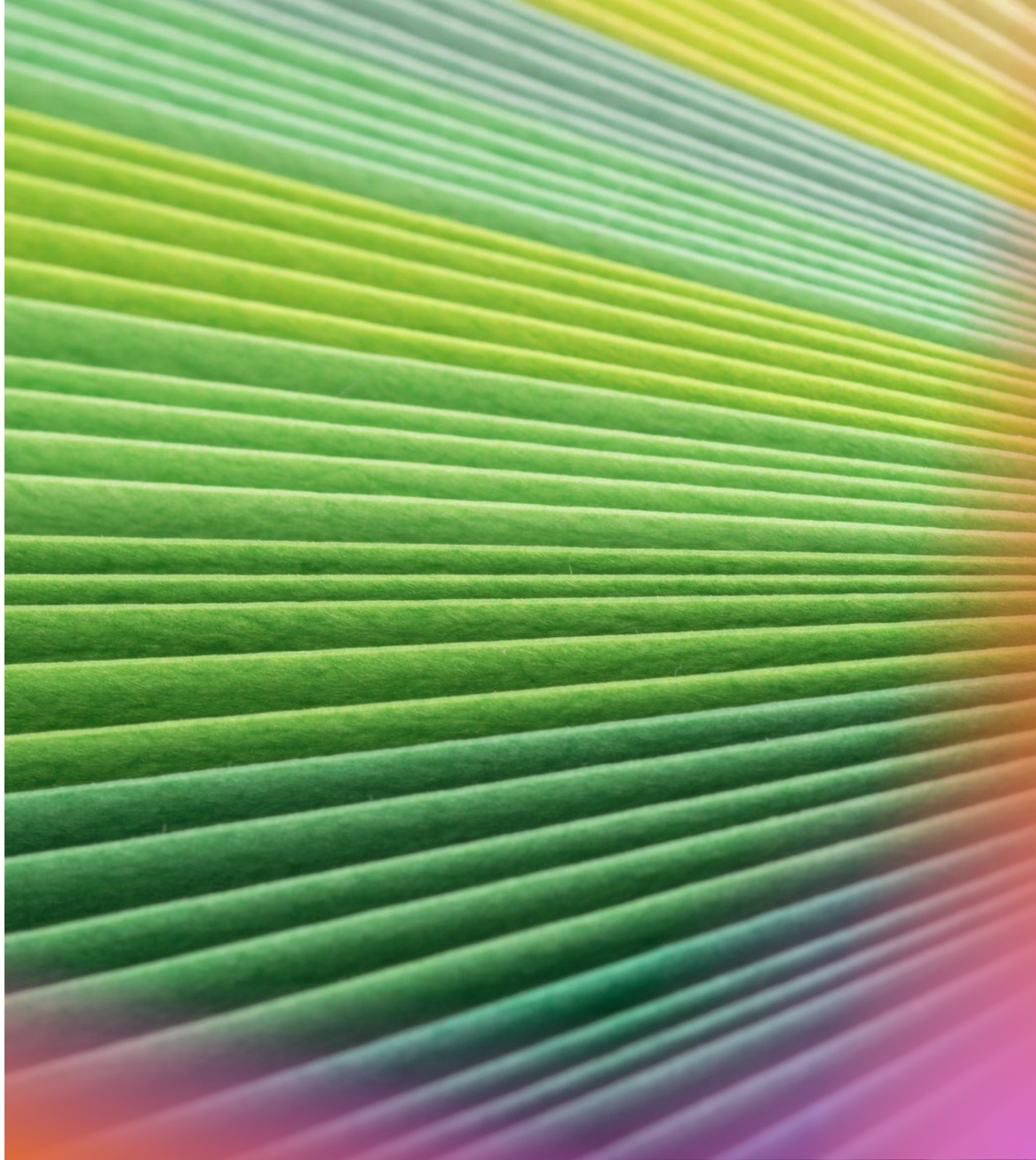
Introduction to Computing and Programming

Multi-Dimensional Arrays, Functions

Recap

- Operations on Arrays
- Examples of Arrays
- 2D Array

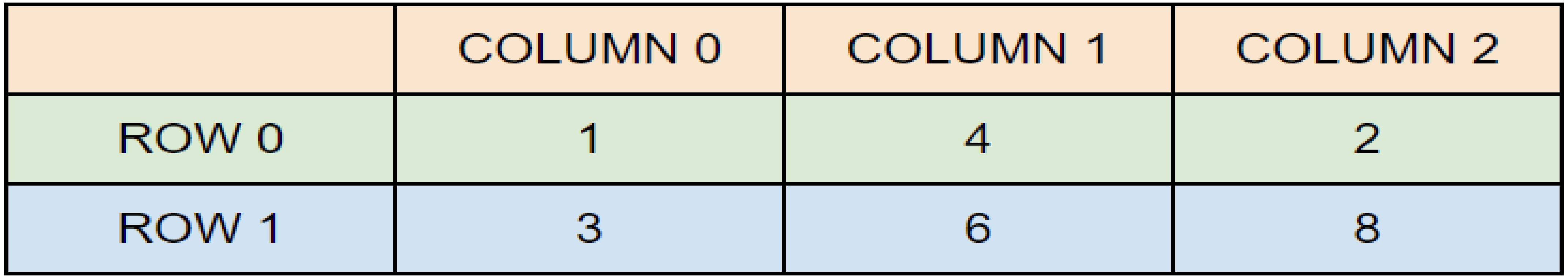


Content

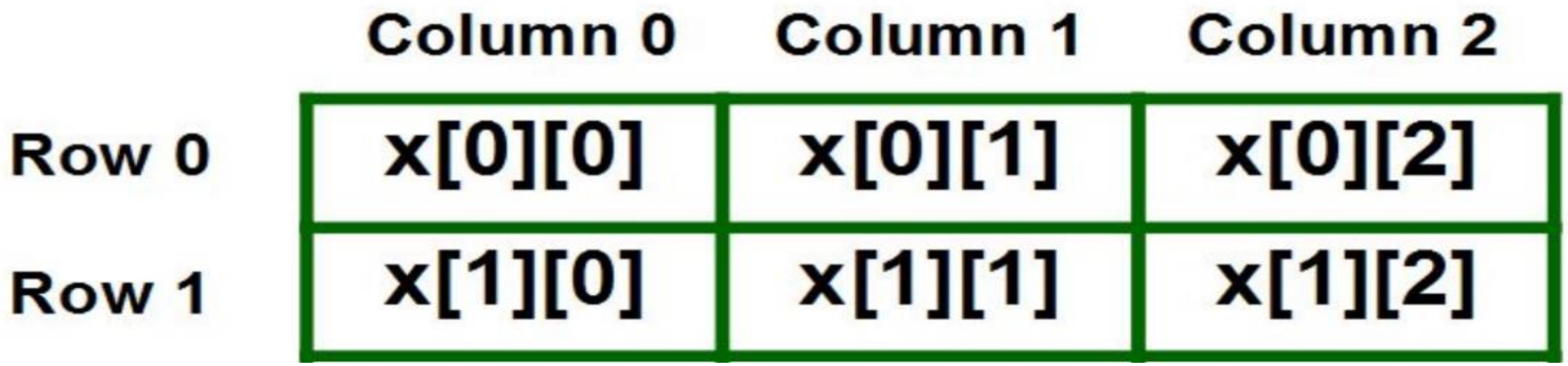
Two-Dimensional Array

- 2D array is also known as a matrix (a table of rows and columns).

- Example:
  - int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };



	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8



	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]

Access the Elements of a 2D Array

- To access an element of a two-dimensional array, you must specify the index number of both the row and column.
- This statement accesses the value of the element in the **first row (0)** and **third column (2)** of the **matrix** array.
- Example

  -
    ```c
    int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

    printf("%d", matrix[0][2]);  // Outputs 2
    ```

Change Elements in a 2D Array

- To change the value of an element, refer to the index number of the element in each of the dimensions:

- The following example will change the value of the element in the **first row (0)** and **first column (0)**:
- Example

  -
    ```
    int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
    matrix[0][0] = 9;

    printf("%d", matrix[0][0]);  // Now outputs 9 instead of 1
    ```

Write a C program to **traverse all the elements in 2D Array**

Example: int arr[3][2] = { { 0, 1 }, { 2, 3 }, { 4, 5 } };

Traversal in 2D Array

```
#include <stdio.h>
int main() {
// Initialize an array with 3 rows and 2 columns
    int arr[3][2] = { { 0, 1 }, { 2, 3 }, { 4, 5 } };
    // Print each array element's value
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            printf("arr[%d][%d]: %d    ", i, j,
arr[i][j]);
        }
      printf("\n"); }
    return 0;}
```

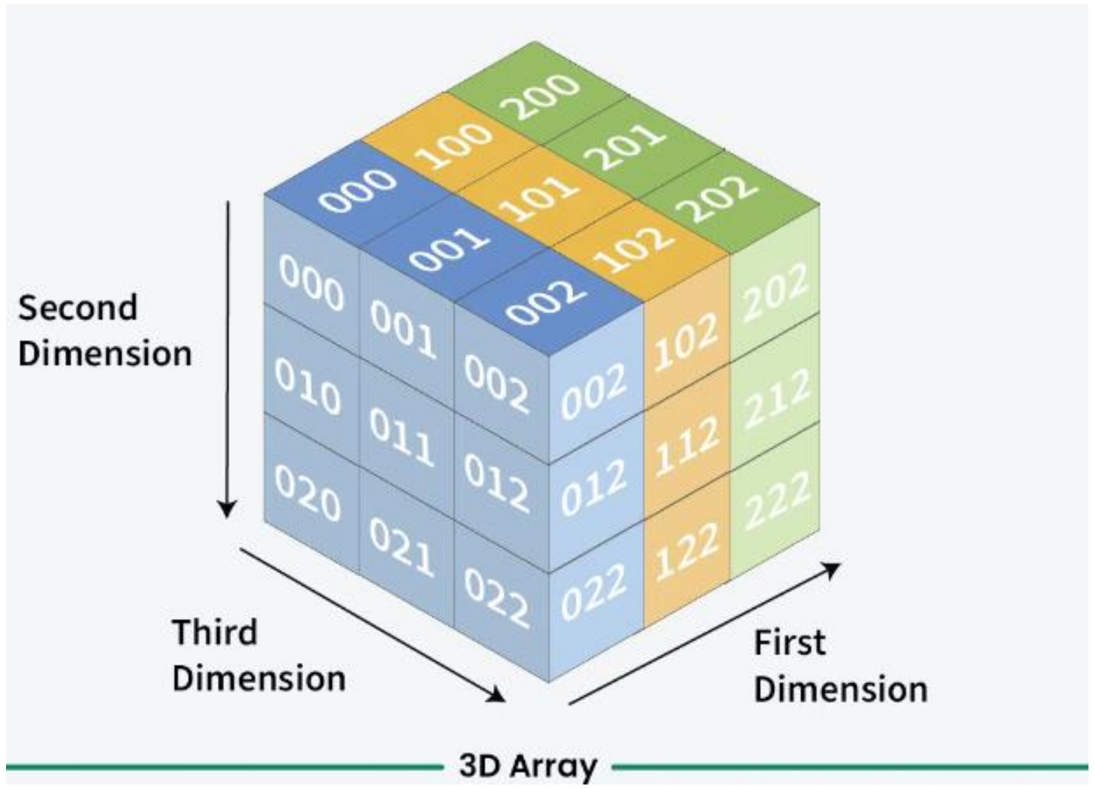
Storing and printing elements at runtime

```
#include <stdio.h>
void main ()
{  int arr[3][3],i,j;
   for (i=0;i<3;i++)
   {    for (j=0;j<3;j++)    {
            printf("Enter a[%d][%d]: ",i,j);
            scanf("%d",&arr[i][j]);
      }    }
   printf("\n printing the elements ....\n");
   for(i=0;i<3;i++)
   {   printf("\n");
      for (j=0;j<3;j++)
      {     printf("%d\t",arr[i][j]);     }    }    }
```

Three-Dimensional (3D) Array in C

- A **Three-Dimensional Array** or **3D** array is a collection of two-dimensional arrays.

- It can be visualized as multiple 2D arrays stacked on top of each other.



Declaration and Initialization

Declaration:

- *type arr_name[x][m][n];*

Initialization:

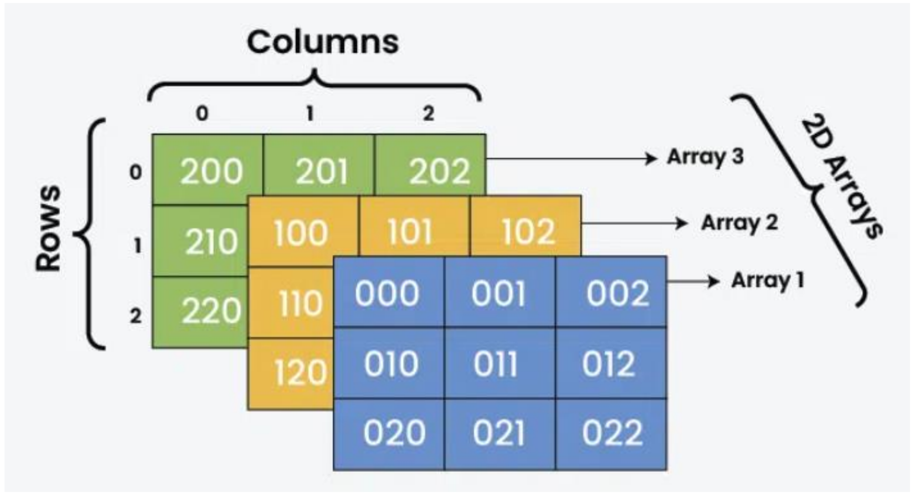
- int arr[2][3][2] = {0, 1, 2, 3, 4, 5, 6, 7 , 8, 9, 10, 11}

Or

- int arr[2][3][2] = { { { 0, 1 }, { 2, 3 }, { 4, 5 } },
- { { 6, 7 }, { 8, 9 }, { 10, 11 } } };

Traversal in 3D array

```
#include <stdio.h>
int main() {
    // Create and Initialize the 3-dimensional array
    int arr[2][3][2] = { { { 1, 1 }, { 2, 3 }, { 4, 5 } }, { { 6, 7 }, { 8, 9 }, { 10, 11 } } };
     for (int i = 0; i < 2; ++i) {// Loop through the depth
          for (int j = 0; j < 3; ++j) {// Loop through the rows of each depth
              for (int k = 0; k < 2; ++k) // Loop through the columns of each row
               printf("arr[%i][%i][%i] = %d   ", i, j, k, arr[i][j][k]);
             printf("\n");}
      printf("\n\n");  }
    return 0;}
```



Advantages and Disadvantages

Advantages:

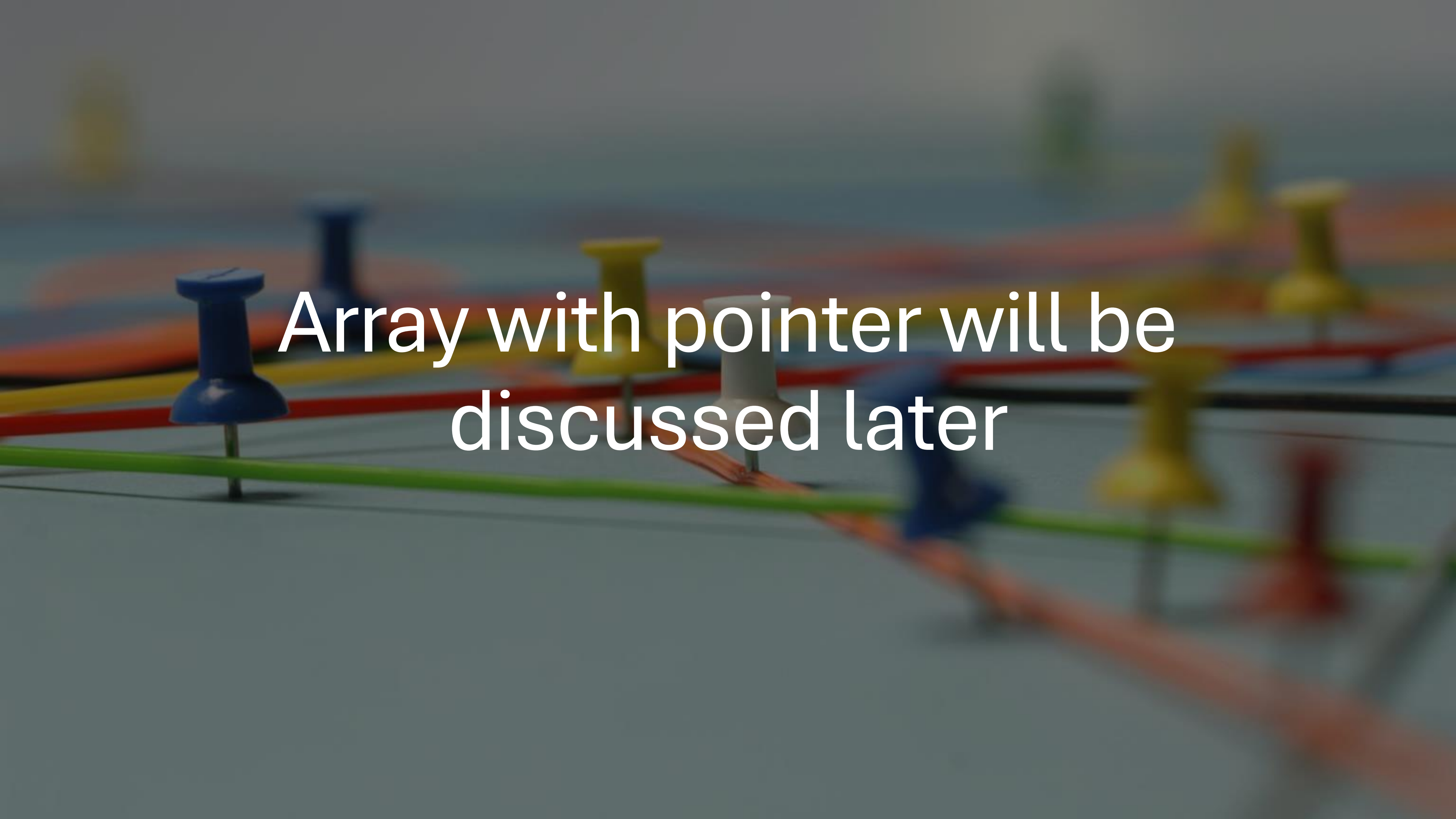
- Fast access to elements.
- Efficient memory usage.

Disadvantages:

- Fixed size (in static arrays).
- Insertion and deletion can be costly.

Use Cases of Arrays

- **Data Storage:**
  - Storing collections of data.
- **Matrix Representation:**
  - 2D arrays for matrices.
- **Buffers and Tables:**
  - Use in graphics, tables, and buffers.



Array with pointer will be discussed later

Some Important announcement

- We will **post the Assignment on Saturday or Sunday** that would be fair with Monday batch as well.
- Next week, **Thursday (26th Sept) class would be of revision class**; **Send all the questions or topics that you want to revise;** Attendance will be given to all the students.
- Till **Function** would be given to Mid-sem; I will upload the **question bank of Array & Function** early next week.
- Will discuss the Mid-sem pattern on Tuesday, 24th Sept.
- We will be taking **graded lab 2 from 7th to 11th Oct**.
- **LASC Tutor**

Functions

What is a function in C?

A program segment that carries out **some specific, well-defined task.**

Example:
1. A function to **add two numbers**
2. A function to **find the largest of n numbers**

A function will carry out its intended task **whenever it is called or invoked**

Can be called **multiple times**

Purpose of Function:

- Modularize code.
- Enhance reusability.
- Improve readability and maintainability

Functions – Characteristics

- Every C program consists of **one or more functions**

- One of these functions must be **called main()**
  - Every C program has at least one function – main() – and all the most trivial programs can define additional functions.
  - You can divide up your code into separate functions
  - How you divide up your code among different functions is up to you, but logically the division is such that **each function performs a specific task**

- **Note** that the execution of a C program always begins by carrying out the instructions in main()
  - Functions call other functions as instructions

Function Declaration

- **Syntax:**

```
return_type function_name(parameters);
```

- **Example:**

```
int add(int a, int b);
```

Function Definition

Syntax:

```
return_type
  function_name(parameters) {
    // body
    return value;
}
```

Example:

```
int add(int a, int b) {
return a + b;
}
```

Function Definition Cont..

- The general **skeleton/syntax** of a function in C is as follows:

```
return_type function_name ( parameter list ) {
// body of the function
}
```

- A **function definition** in C consists of:
  - a function header and
  - a function body
- **Function Declaration**:
  - Tells the compiler about a function's name, return type, and parameters
  - A function definition provides the actual body of the function

Functions - Components

- **Return Type:**
  - A function may **return a value**
  - The return_type is the data type of the value the function returns.
  - Some functions perform the desired operations without returning a value.
    - In this case, the **return_type** is the keyword **void**.
- **Function Name:**
  - Actual name of the function
  - The function name and the parameter list together constitute the **function signature**.

Functions – Components Cont..

Parameters:

-A parameter is like a placeholder.

-When a function is invoked, you pass a value to the parameter.

-This value is referred to as actual parameter or argument.

-The parameter list refers to the type, order, and number of the parameters of a function.

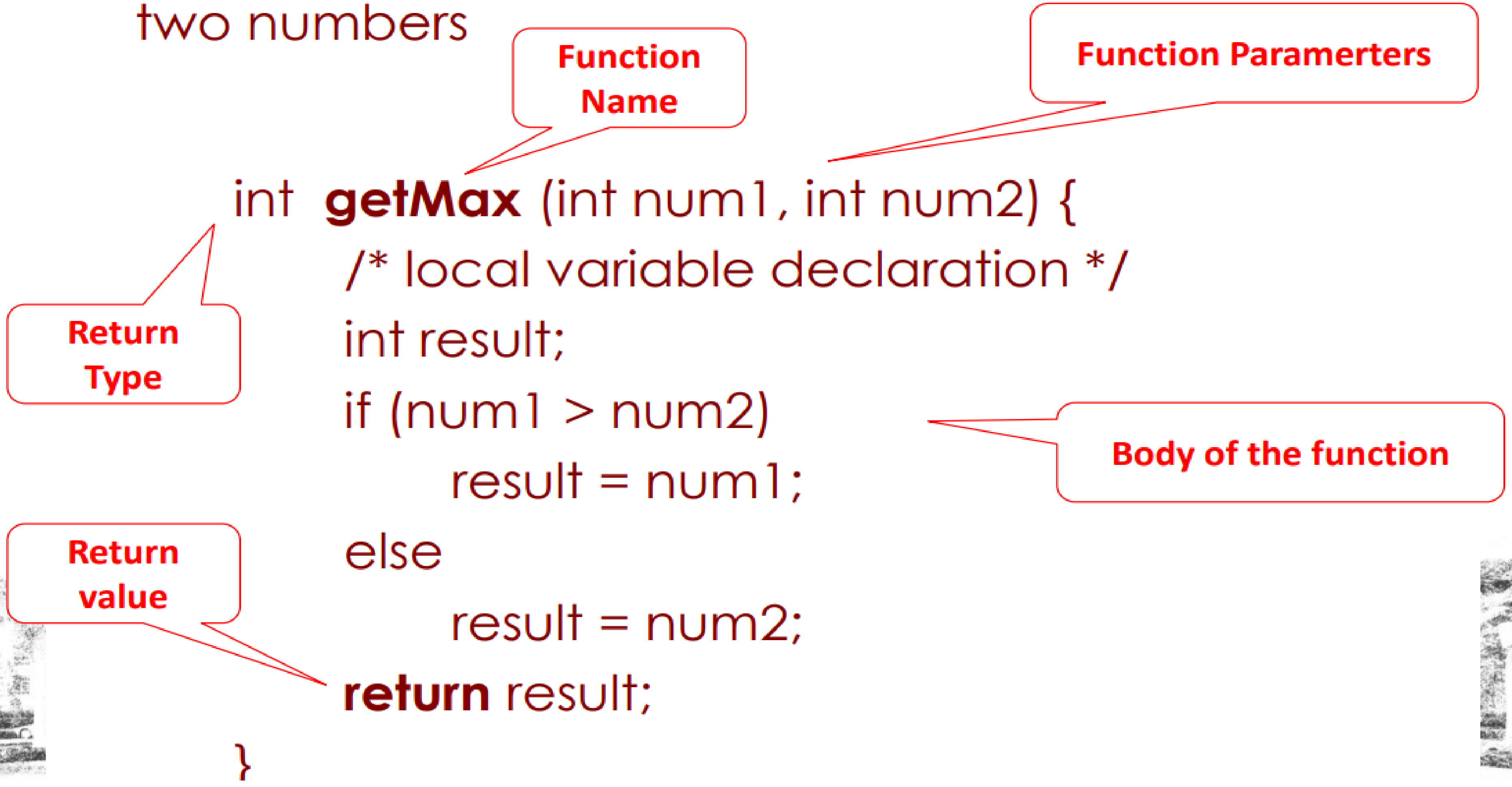
-Parameters are optional; this means that a function may contain no parameters.

Function Body: The function body contains a collection of statements

that define what the function does.

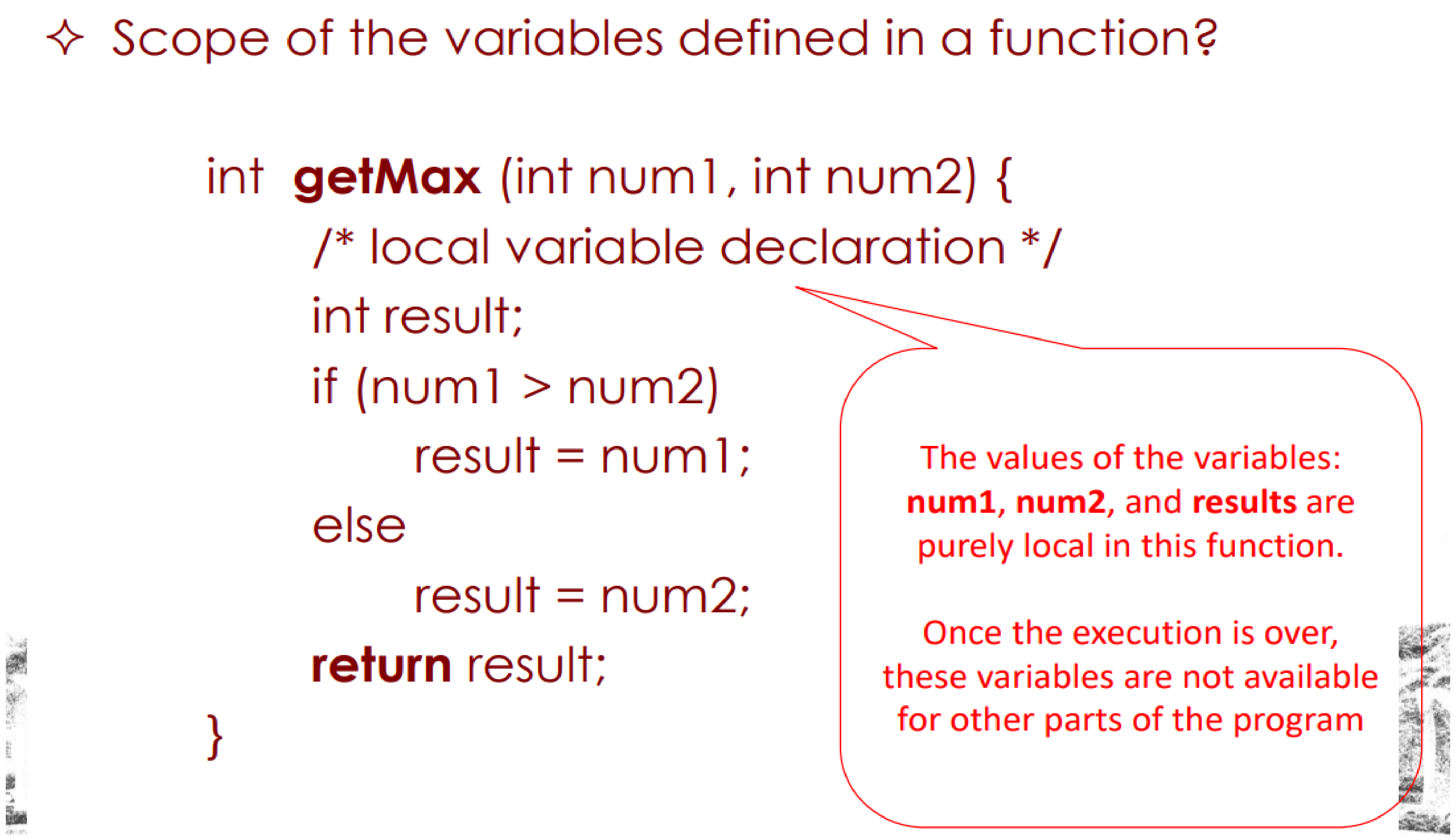
Function an Example:

- The following function returns the max between two numbers



```
int  getMax (int num1, int num2) {
     /* local variable declaration */
     int result;
     if (num1 > num2)
          result = num1;
     else
          result = num2;
     return result;
}
```

Scope of the variables

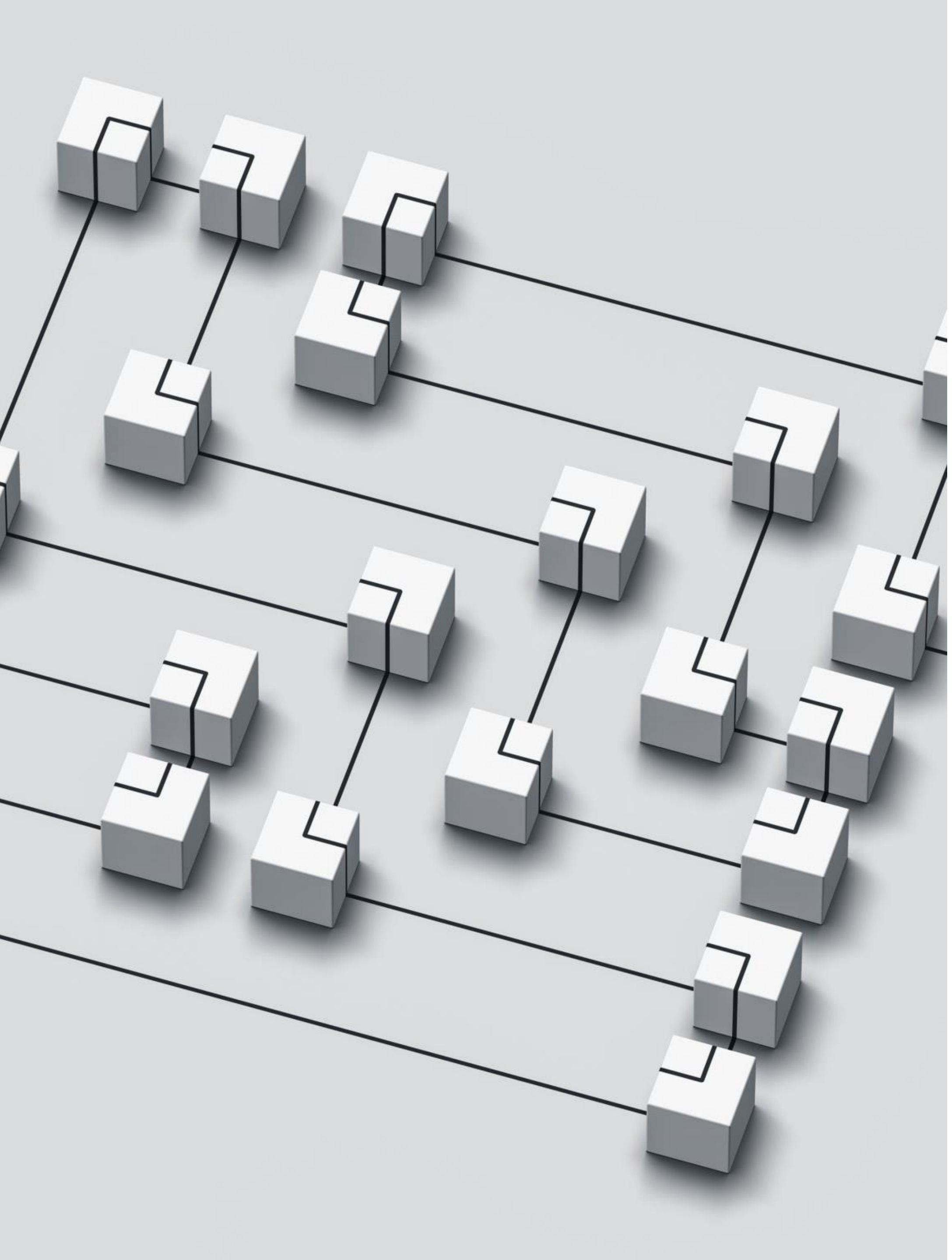


✧ Scope of the variables defined in a function?

```
int  getMax (int num1, int num2) {
     /* local variable declaration */
     int result;
     if (num1 > num2)
          result = num1;
     else
          result = num2;
     return result;
}
```

The values of the variables: **num1**, **num2**, and **results** are purely local in this function.

Once the execution is over, these variables are not available for other parts of the program



Upcoming Slides

- More about Functions
- Example of Functions
- Functions with Arrays
- Recursion
- Macro & Inline Function