**Tishk International University**
Faculty of Applied Science
Information Technology Department

# *Arrays*

Lecture 1

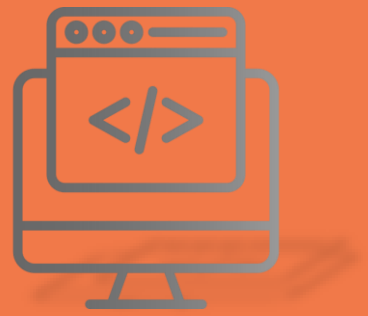
Spring 2026

Course Code: IT118

Grade 1

**Islam Abdulazeez**
islam.abdulaziz@tiu.edu.iq

April 6, 2026



## Programming II

# Outlines

- ✓ Introduction to Arrays
- ✓ Declaration and Initialization
- ✓ Accessing and Processing Arrays
- ✓ Multidimensional Arrays

# Learning Outcomes

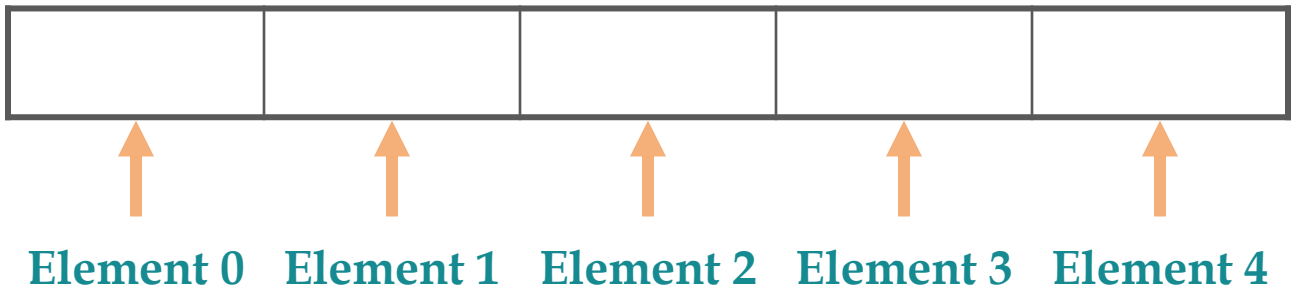
- **At the end of today's session, you will be able to:**

  - ✓ Define arrays and explain their structure and indexing.
  - ✓ Use arrays in C++ for storing and accessing data.
  - ✓ Identify and correct common array errors.
  - ✓ Develop programs using arrays to solve problems.

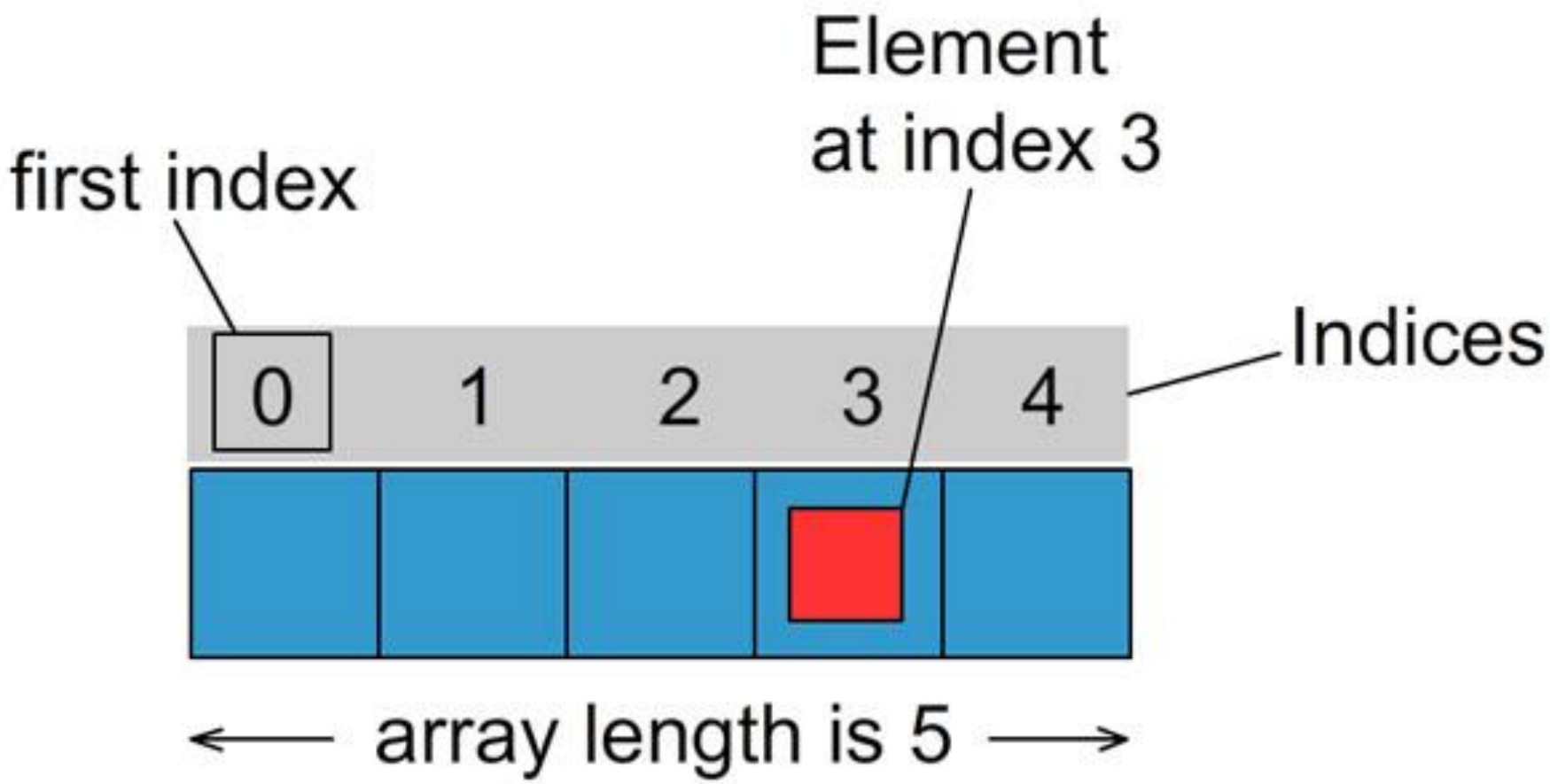
# What is an Array?

- **Array**: A variable that can store multiple values of the same type
- Values are stored in adjacent memory locations
- Declared using [] operator
- Example: **int A[5];**

This declaration allocates the following memory locations



# Indexes


Element
at index 3
first index
Indices
0
1
2
3
4
array length is 5

# Defining Array

```
Datatype arrayName[arraySize];

int numbers[5];    //holds 5 integer cells
```

- int is the data type of the array elements
- numbers is the name of the array
- 5 is the number of elements.

Another Example:

```
double volumes[10];    //holds 10 double cells
```

# Array Size Declarators

| Array declaration | Number of elements | Size of each element | Size of the array |
|---|---|---|---|
| `char letter[26];` | 26 | 1 byte | 26 bytes |
| `short ring[100];` | 100 | 2 bytes | 200 bytes |
| `int mile[84];` | 84 | 4 bytes | 336 bytes |
| `float temp[12];` | 12 | 4 bytes | 48 bytes |
| `double distance[1000];` | 1000 | 8 bytes | 8,000 bytes |

# Initializing Arrays

- Arrays in C++ can be initialized in several ways:

1. Initialization with Assignment Statements: Individual array elements can be initialized during program execution using assignment statements.

```
A[0] = 79;
A[1] = 82;
```

2. Initialization at Array Definition with an Initialization List:

```
int A[5] = {79,82,91,77,84};
```

# Initializing Arrays

- Special Cases for Initialization Lists:

```
int A[5] = {0}; //all have zero value
int A[5] = {4}; //only the first element has value 4, others are zero
```

# Implicit Array Sizing

- Can determine array size by the size of the initialization list

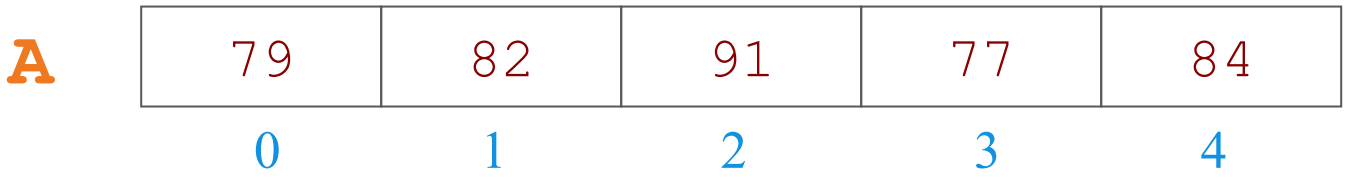
```
int quizzes[]={12, 17, 15, 11};
```

Must use either **array size declarator** or **initialization list** when array is defined.

# Accessing Array Elements

- Array elements can be treated like regular variables and are accessed using the array name and index.

```
int A[5] = {79,82,91,77,84};
```




```
cout << A[0];  //Prints 79
cout << A[2];  //Prints 91
```

# Inputting Array Contents

- **cin** can be used to input values from keyboard and store these values into an array element.

```
int A[5];  // Define 5-cells array
cout << "Enter first number ";
cin >> A[0];
cout << "Enter second number ";
cin >> A[1];
```

# Processing Array Contents

- Array elements can be treated just like regular variables of the same type as the array.
- You can perform arithmetic operations using array elements.
- Array elements can be used in relational expressions, like compare elements.
- They can also be used in logical expressions, like AND, OR, and NOT operations.

```
int numbers[] = {5, 2, 3};

int sum = numbers[0] + numbers[1] + numbers[2];
int product = numbers[0] * numbers[1] * numbers[2];

if (numbers[0] <= 0) {
    cout << "Element 1 is less than or equal to 0";
}
else {
    cout << "Element 1 is greater than 0";
}
```

# Displaying The Size of Array

```
int num[] = {21,45,11,8};

int size = sizeof(num) / sizeof(num[0]);

cout << size;   //4
```

# Array Subscripts

- Array subscript (index) can be an **integer constant**, **integer variable**, or **integer expression**.

```
// Declare an array
int arr[] = { 10, 12, 0, 4, 3 };

// Accessing array elements using integer constants
cout << "Element at index 0: " << arr[0] << endl; // Output: 10
cout << "Element at index 2: " << arr[2] << endl; // Output: 0

// Accessing array elements using integer variables
int index = 3;
cout << "Element at index " << index << ": " << arr[index] << endl; // Output: 4

// Accessing array elements using integer expressions
int i = 1, j = 2;
cout << "Element at index " << i + j << ": " << arr[i + j] << endl; // Output: 4
```

# Displaying All Array Elements

- To display each element of an array, use a loop

```
float A[5] = { 5.76, 8.1, 0.5, 6.1, 3.5 };

int size = sizeof(A) / sizeof(A[0]);

for (int i = 0; i < size; i++) {
    cout << A[i] << endl;
}
```

# Sample Input and Output Program of an Array

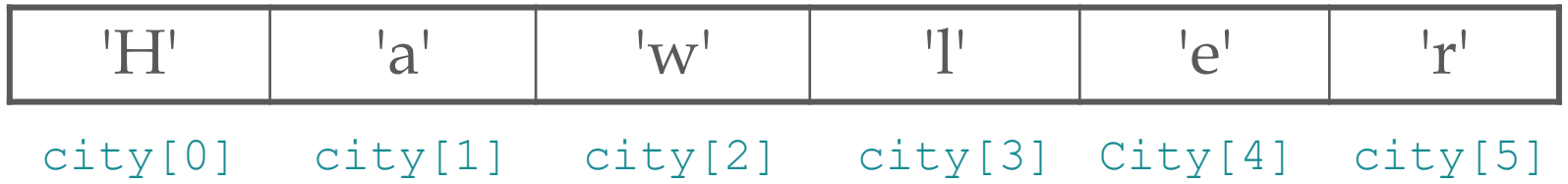
```
int A[4];
int size = sizeof(A) / sizeof(A[0]);

for (int i = 0; i < size; i++) {
    cout << "Enter number to array: ";
    cin >> A[i];
}

for (int i = 0; i < size; i++) {
    cout << A[i] << endl;
}
```

```
Enter number to array: 5
Enter number to array: 2
Enter number to array: 3
Enter number to array: 7
5
2
3
7
```

# Strings and string Objects

- String is a special type of array of characters.
- It can be processed using array name
  - Entire string at once, or
  - One element at a time by using a subscript

```
string city = "Hawler";
```



# Strings and string Objects

```
string city = "Hawler";

for (int i = 0; i < city.length(); i++) {
    cout << city[i] << endl;
}
```

```
H
a
w
l
e
r
```

**Note:** length() function returns how many characters are in a string.

# Example

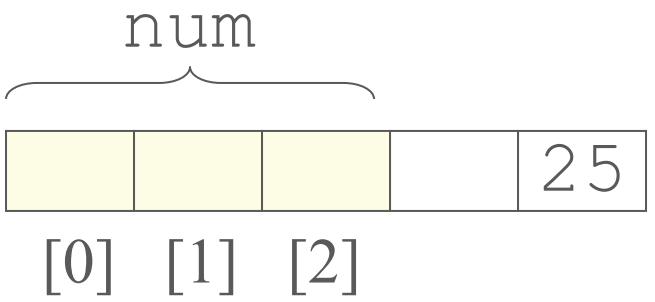
```
const int NUM_MONTHS = 12;

string name[NUM_MONTHS] = {"January", "February", "March", "April", "May",
    "June", "July", "August", "September", "October",
    "November", "December"};

int days[NUM_MONTHS] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

for (int month = 0; month < NUM_MONTHS; month++) {
    cout << name[month] << " has " << days[month] << " days.\n";
}
```

```
January has 31 days.
February has 28 days.
March has 31 days.
April has 30 days.
May has 31 days.
June has 30 days.
July has 31 days.
August has 31 days.
September has 30 days.
October has 31 days.
November has 30 days.
December has 31 days.
```

# NOTE 1: No Bounds Checking

- C++ does not check whether an array subscript is in range.
- An invalid array subscript can cause program to overwrite other memory locations

```
int num[3]; // composed of => num[0], num[1], num[2]
cout << sizeof(num) / sizeof(num[0]) << endl; // 3

int i = 4;
num[i] = 25; // we don't have num[4]

cout << sizeof(num) / sizeof(num[0]) << endl; // 3
```



# NOTE 2: Using Increment and Decrement

- When using ++ and -- operators, don't confuse the element with the subscript

```
A[i]++; // adds 1 to A[i]
A[i++]; // increments i, but has no effect on A

int score[5] = {7, 8, 9, 10, 11};
++score[2]; // Pre-increment operation on the value in score[2]
score[4]++; // Post-increment operation on the value in score[4]
```

# NOTE 3: Copying One Array to Another

```
int A1[5]= {1,2,3,4,5};

int A2[5];
```

- Cannot copy with an assignment statement:

```
A1 = A2     // Not allowed
```

- ✓ But we can copy with an assignment statement inside a loop:

```
int A1[] = { 1, 2, 3, 4, 5 };
int A2[5];

int sizeA1 = sizeof(A1) / sizeof(A1[0]);

for (int i = 0; i < sizeA1; i++) {
    A2[i] = A1[i];
}
```

# Search Inside an Array

**Q.** Write a program to search for an input inserted by user in the array.

```
int myArray[] = { 3, 4, 7, 1, 9 };
int size = sizeof(myArray) / sizeof(myArray[0]);
int user;
bool found = false;

cout << "Search for a number: ";
cin >> user; 2

for (int i = 0; i < size; i++) {
    if (myArray[i] == user) {
        found = true;
        break;
    }
}

if (found) {
    cout << "Number found.";
}
else {
    cout << "Number not found.";
}
```

# Vowel Counter Application

**Note:** tolower() function converts a character to its lowercase version.

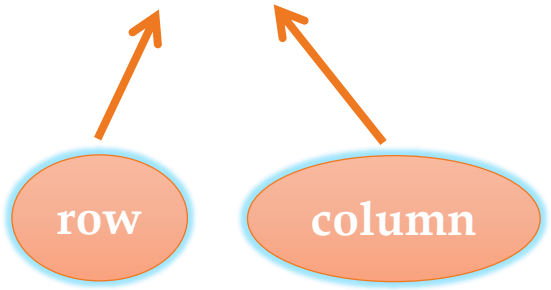
```
string myText;
int counter = 0;
char ch;

cout << "Input any sentences to count number of Vowels:" << endl;
getline(cin, myText);

for (int i = 0; i < myText.length(); i++) {
    ch = tolower(myText[i]);

    if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
        counter++;
    }
}

cout << "There are " << counter << " vowels in the Text.";
```

# Two-Dimensional (2D) Array

- A 2D array is an array of arrays, used to store data in rows and columns much like a table or matrix.

- dataType arrayName[rows][columns];

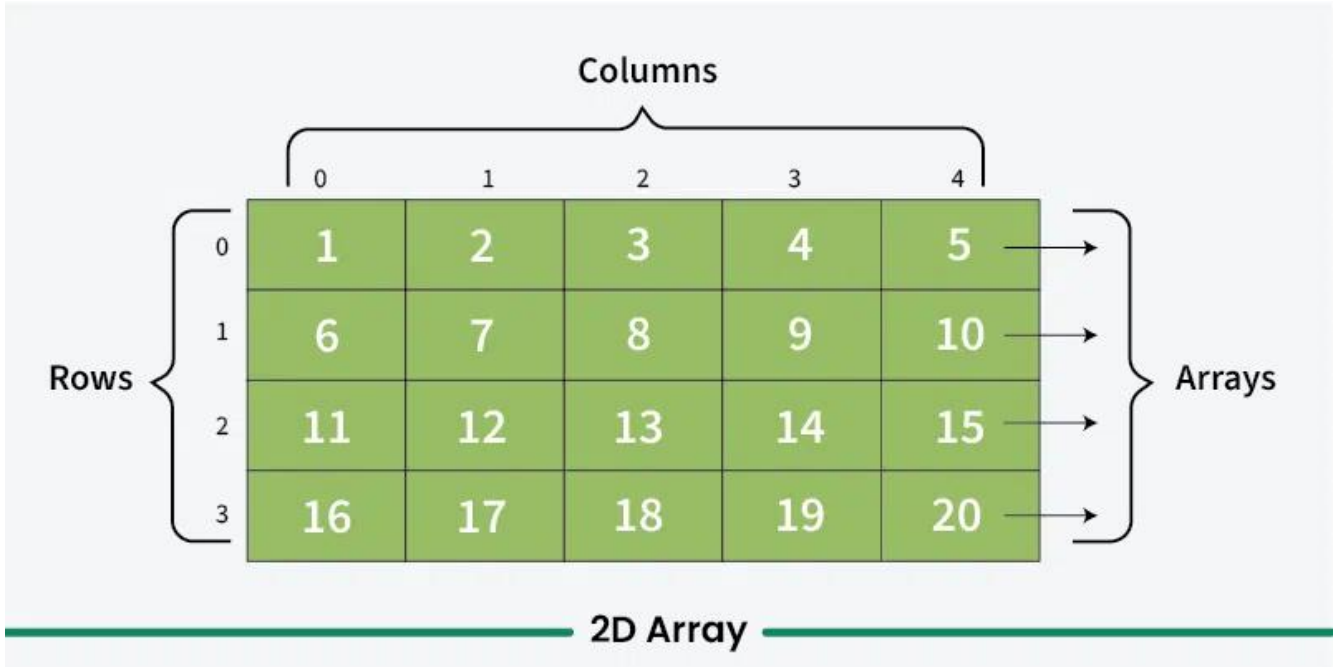
```
int Arr[4][4]
```




| | Col1 | Col2 | Col3 | Col4 | .... |
|---|---|---|---|---|---|
| Row1 | Arr[0][0] | Arr[0][1] | Arr[0][2] | Arr[0][3] | |
| Row2 | Arr[1][0] | Arr[1][1] | Arr[1][2] | Arr[1][3] | |
| Row3 | Arr[2][0] | Arr[2][1] | Arr[2][2] | Arr[2][3] | |
| Row4 | Arr[3][0] | Arr[3][1] | Arr[3][2] | Arr[3][3] | |
| ⋮ | | | | | |

# Two-Dimensional (2D) Array

```
int arr[4][5] = {
{1,  2,  3,  4,  5},
{6,  7,  8,  9, 10},
{11, 12, 13, 14, 15},
{16, 17, 18, 19, 20}
};
```



# Two-Dimensional (2D) Array

```
int arr[2][3]= { {2, -5, 6},{4, 0, 9} };

int arr[2][3]= { {2, -5, 6},
                 {4, 0, 9}
                 };
```

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 2 | -5 | 6 |
| 1 | 4 | 0 | 9 |

# Two-Dimensional (2D) Array

- Example

```
int arr[2][3] = { {2, -5, 6}, {4, 0, 9} };

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        cout << arr[i][j] << "\t";
    }
    cout << endl;
}
```

# Two-Dimensional (2D) Array - Input and print values

```
int A[2][3];

// Input values
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        cout << "Enter value for row " << i + 1 << " column " << j + 1 << ": ";
        cin >> A[i][j];
    }
}
// Display values
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        cout << A[i][j] << "\t";
    }
    cout << endl;
}
```

# Example

- Create a C++ program that prompts the user to enter elements into a 2D array (row=3, column=4). Then, find and print the maximum element in the array.

```
int A[3][4];

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        cout << "Input numbers: ";
        cin >> A[i][j];
    }
}

int max = A[0][0];

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        if (A[i][j] > max) {
            max = A[i][j];
        }
    }
}
cout << "Max = " << max;
```

# 3-Dimensional Array

- A 3-dimensional array (3D array) is an array that stores data in three dimensions instead of one or two.

```
int A[2][3][4];
```

- This means:
  - 2 “blocks” (or layers)
  - Each block has 3 rows
  - Each row has 4 columns

- Think of it like this:

```
Layer 0:
[] [] [] []
[] [] [] []
[] [] [] []

Layer 1:
[] [] [] []
[] [] [] []
[] [] [] []
```

# 3-Dimensional Array

- Example

```
int A[2][3][4] = {
{
      {1, 2, 3, 4},
      {5, 6, 7, 8},
      {9, 10, 11, 12}
},
{
      {13, 14, 15, 16},
      {17, 18, 19, 20},
      {21, 22, 23, 24}
}
};
```

# 3-Dimensional Array

- Declare 3-dimensional array and ask user to input the values. After that print the array.

```
int arr[2][3][4];

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        for (int k = 0; k < 4; k++) {
            cout << "Input the number " << endl;
            cin >> arr[i][j][k];
        }
    }
}

// Print the array
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        for (int k = 0; k < 4; k++) {
            cout << arr[i][j][k] << " ";
        }
        cout << endl;
    }
    cout << endl;
}
```

# Activities and Next Lecture's Topic

## Activities

- Review this lecture note
- Practice

## Next Lecture's Topic

- Functions

# References


- Gaddis, T. (2014). Starting out with C++: Early objects (7th ed.). Pearson Education.
- GeeksforGeeks. (2025, November 14). Multidimensional arrays in C. Multidimensional Arrays in C
- Sneh, & Kurup, M. (2025, June 24). Mastering 2D arrays in C++: A comprehensive guide. DigitalOcean. https://www.digitalocean.com/community/tutorials/two-dimensional-array-in-c-plus-plus


# Thank You!