# Introduction to Computing and Programming

## Arrays

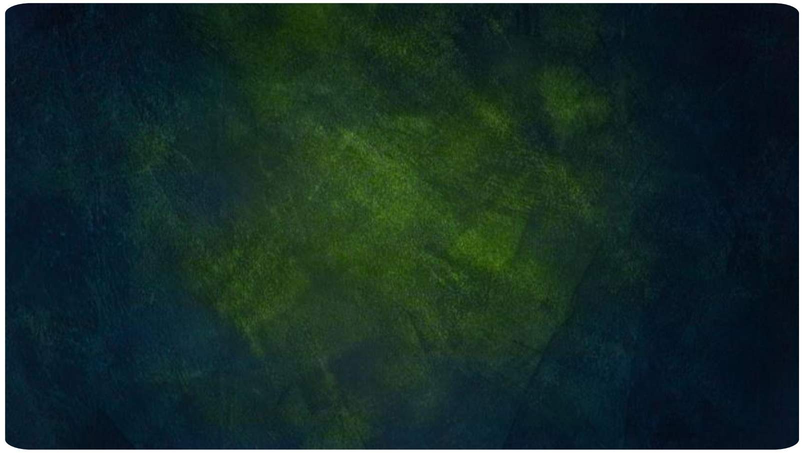
# Recap

Some more Exercise on Loops

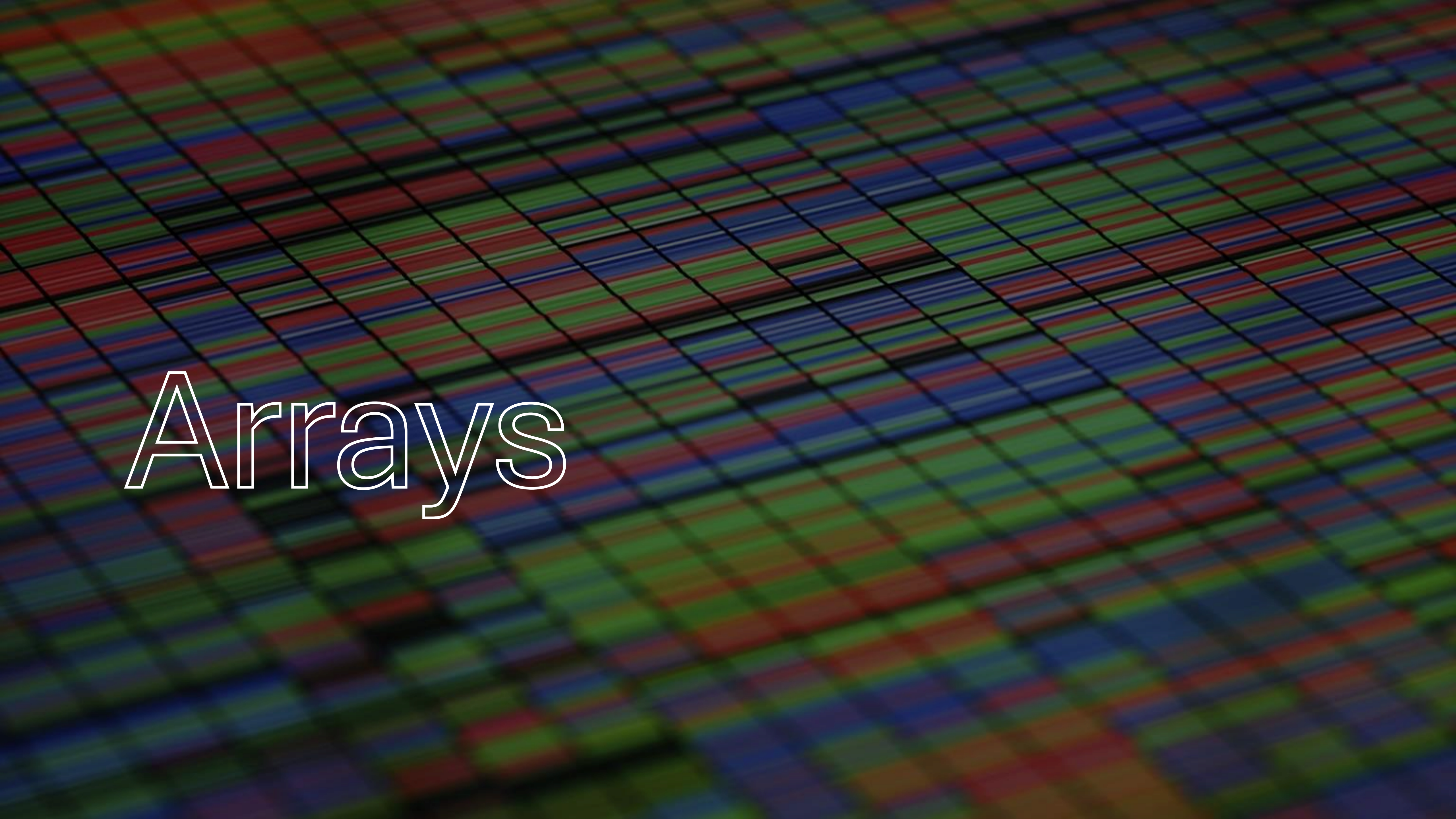
Arrays

# Content

- Some more examples of Arrays
- Types of Arrays: - Two dimensional
- Operations of Arrays
- **Discussion on Quiz 1 Solutions, Lab Assignments rules**



# Arrays

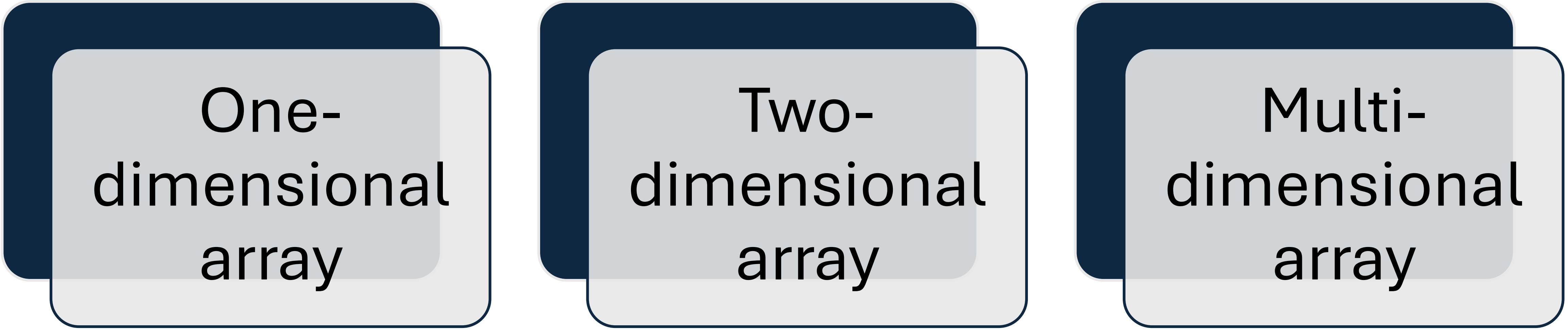


# Introduction to Arrays

An array is **a collection of elements of the same type** that are referenced by a common name.

Called as derived data type.

All the elements of an array occupy a set of contiguous memory locations.



One-dimensional array

Two-dimensional array

Multi-dimensional array

# 1-D Array Declaration & Initialization

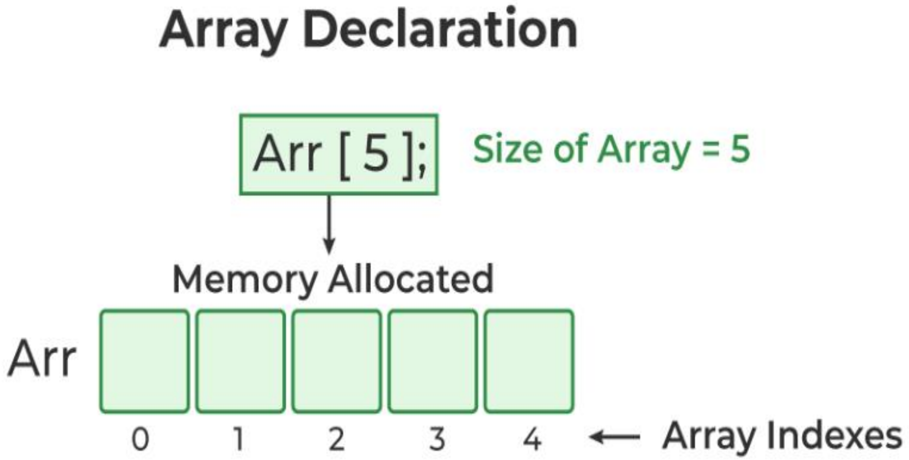
- **Syntax**
  - data_type array_name[array_size];

- **Initialization with Declaration**

Syntax:

  - *data_type array_name* [*size*] = {value1, value2, … valueN};

- Array Initialization with Declaration without Size
  - data_type array_name[] = {1,2,3,4,5};
  - The size of the above arrays is 5 which is automatically deduced by the compiler.



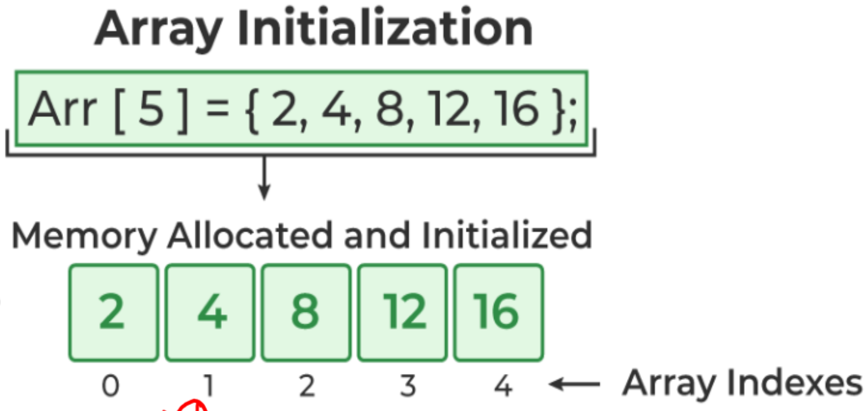
**Array Declaration**

Arr [ 5 ]; Size of Array = 5

Memory Allocated

Arr

0 1 2 3 4 ← Array Indexes



**Array Initialization**

Arr [ 5 ] = { 2, 4, 8, 12, 16 };

Memory Allocated and Initialized

| Arr | 2 | 4 | 8 | 12 | 16 |
|---|---|---|---|---|---|
|  | 0 | 1 | 2 | 3 | 4 ← Array Indexes |

# Array Initialization after Declaration (Using Loops)

- We initialize the array after the declaration by assigning the initial value to each element individually.

- We can use **for loop, while loop, or do-while loop** to assign the value to each element of the array.
- Example:

```
for (int i = 0; i < N; i++)
{
    array_name[i] =
valuei;
}
```

array_name[5]=value$_5$;

# Example on Array Initialization

```
#include <stdio.h>
int main()
{
int arr[5] = { 10, 20, 30, 40, 50 }; // array initialization using initialiser list
int arr1[] = { 1, 2, 3, 4, 5 }; // array initialization using initializer list without specifying size
float arr2[5]; // array initialization using for loop
for (int i = 0; i < 5; i++) {
        arr2[i] = (float)i * 2.1;}
    return 0;
}
```

# Array Elements in Memory

- 16 bytes get immediately reserved in memory, 2 bytes each for the 8 integers .
- The array is not being initialized; all eight values present in it would be garbage values.
- Whatever be the initial values, all the array elements would always be present in contiguous memory locations.

int arr[8] ;

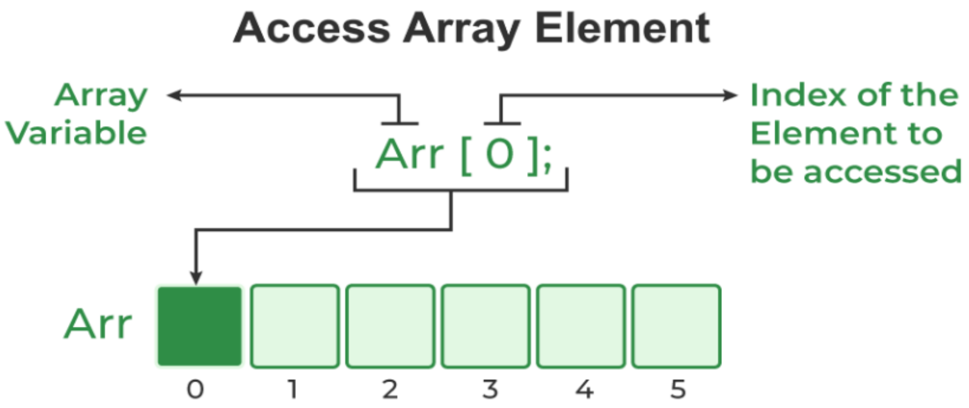
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 12 | 34 | 66 | -45 | 23 | 346 | 77 | 90 | GP |
| 65508 | 65510 | 65512 | 65514 | 65516 | 65518 | 65520 | 65522 | |

# Bound checking in Array

- No check` to see if the subscript used for an array exceeds the size of the array.
- Exceeded data will simply be placed in memory outside the array; probably on top of other data, or on the program itself.
- This will lead to unpredictable results
- No error message to warn

```
main( )
{
int num[40], i ;
for ( i = 0 ; i <= 100 ; i++ )
num[i] = i ;
}
```

# Access Array Elements



- We can access any element using the array subscript operator [ ] and the index value i of the element.
  - array_name [index];
- Indexing in the array always **starts with 0 and the last element is at N – 1** where N is the number of elements in the array.

```
int arr[5] = { 15, 25, 35, 45, 55 }; // array declaration and initialization

printf("Element at arr[2]: %d\n", arr[2]); // accessing element at index 2 i.e 3rd element
```

# Basic Array Operations

Following are the basic Array operations:

1. **Traverse** − Print each element in the array one by one.

2. **Insertion** − At the specified index, adds an element.

3. **Deletion** − The element at the specified index is deleted.

4. **Search** − Uses the provided index or the value to search for an element.

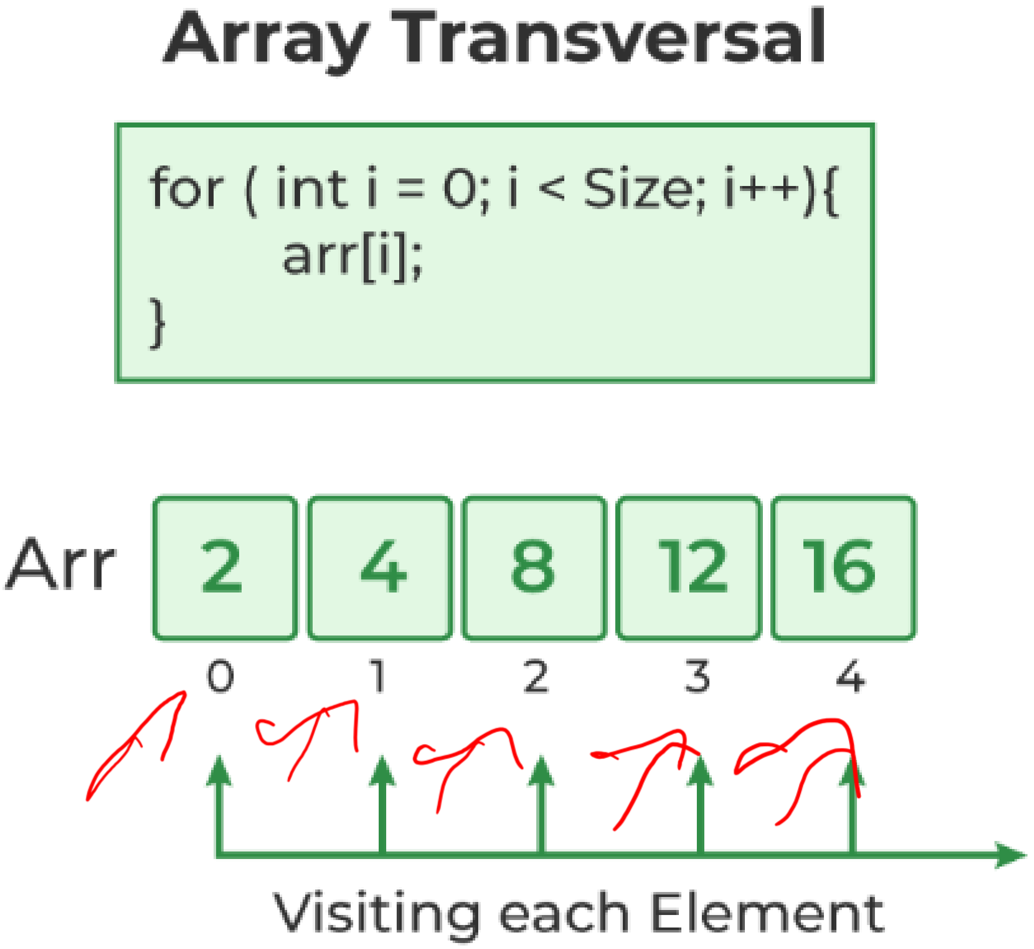
5. **Update** − The element at the specified index is updated.

# Array Traversal

- Traversal is the process in which we visit ever element of the data structure.
- **Loops** to iterate through each element of the array.

- Array Traversal using for Loop

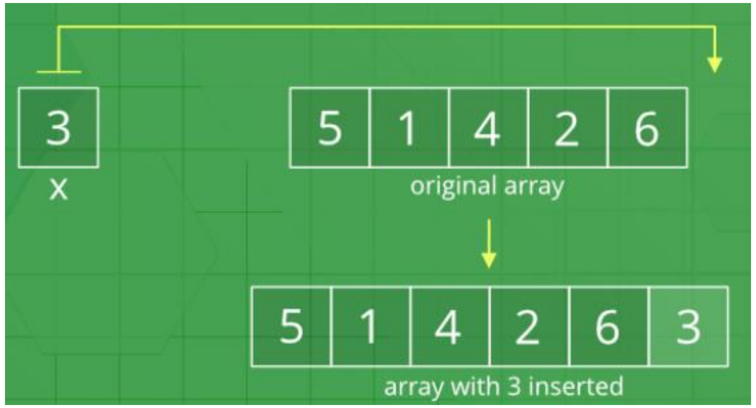
```
for (int i = 0; i < N; i++) {
    array_name[i];
}
```



Array Transversal

```
for ( int i = 0; i < Size; i++){
        arr[i];
}
```

# Array Operation: **Insert an element** at the end of the array



3

x

| 5 | 1 | 4 | 2 | 6 |
|---|---|---|---|---|

original array

| 5 | 1 | 4 | 2 | 6 | 3 |
|---|---|---|---|---|---|

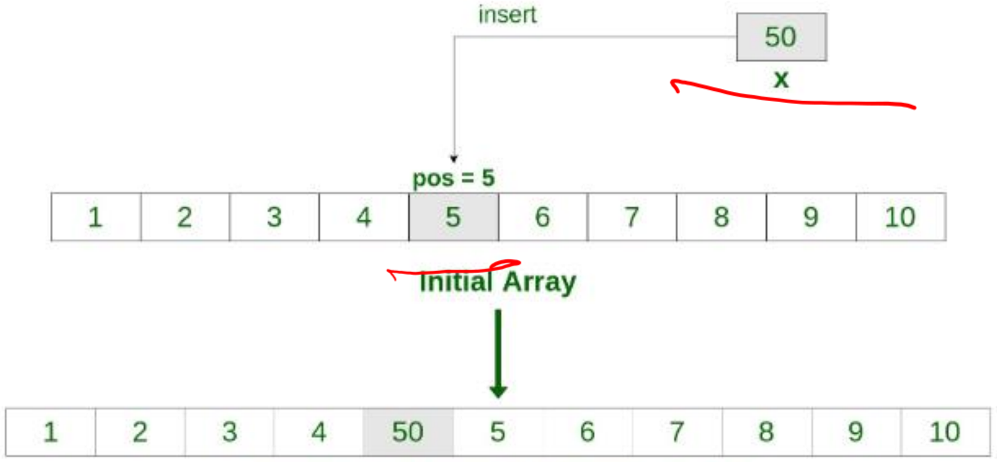
array with 3 inserted

20/2

```
#include <stdio.h>
int main()
{
    int arr[10] = { 12, 16, 20, 40, 50, 70 };
    int capacity = sizeof(arr) / sizeof(arr[0]);
    int n = 6;
    int i, key = 26;   → 8
    printf("\n Before Insertion: ");
    for (i = 0; i < n; i++)
        printf("%d  ", arr[i]);
```

```
if (n >= capacity)
        {printf("\nElement cannot be inserted");}
  else{
     arr[n] = key;
     n = n+1;}
   printf("\n After Insertion: ");
   for (i = 0; i < n; i++)
       printf("%d  ", arr[i]);
   return 0;
}
```

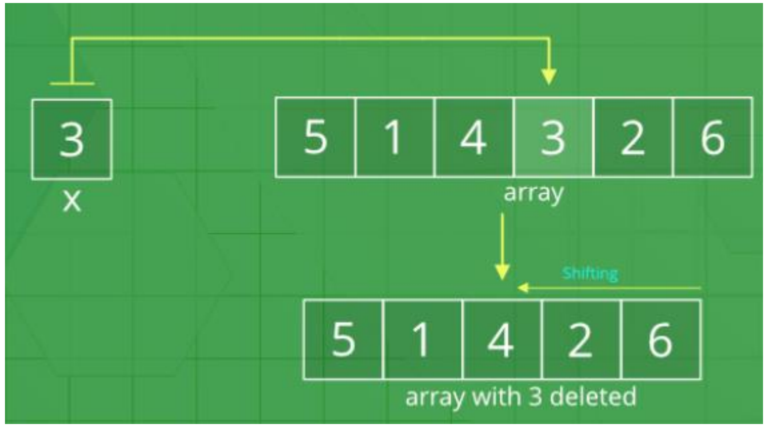
# Array Operation: Insert an element at a position pos in the array



Array with X inserted at position pos

```
int main()
{
   int arr[15] = { 2, 4, 1, 8, 5 };
   int n = 5;
   printf("Before insertion : ");
   for (int i = 0; i < n; i++){
      printf("%d ", arr[i]);}
   printf("\n");
   int x = 10, pos = 2;
  for (int i = n - 1; i >= pos; i--){ arr[i + 1] = arr[i];}
   arr[pos] = x;
   n++;
   printf("After insertion : ");
   for (int i = 0; i < n; i++) { printf("%d ", arr[i]);}
   return 0;
}
```

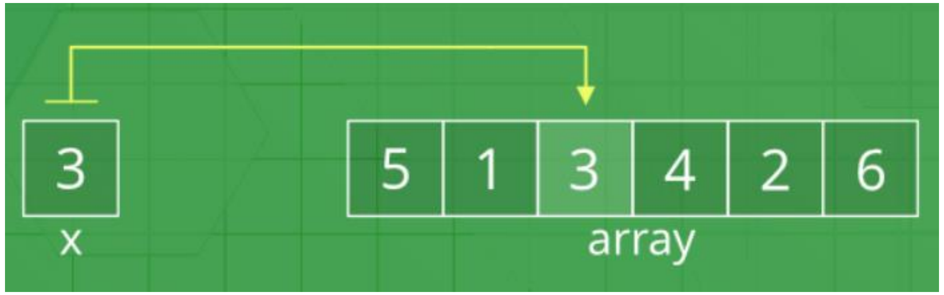
# Array Operation:
# **Delete an element**



```
int main()
{
    int i;
    int arr[] = { 10, 50, 30, 40, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 60;
    printf("Array before deletion\n");
    for (i = 0; i < n; i++){
        printf("%d  ", arr[i]);}
    int pos = -1;
    for (int i = 0; i < n; i++)
    { if (arr[i] == key){
            pos = i;}}
```

```
if (pos == -1) {
        printf("\nElement not found");
    }
//Shift elements to the left from the position to delete
    else{
     for (int i = pos; i < n - 1; i++){
        arr[i] = arr[i + 1];}
        n = n-1;} // Reduce the size of the array by 1

    printf("\nArray after deletion\n");
    for (i = 0; i < n; i++){
        printf("%d  ", arr[i]);}
    return 0; }
```



# Array Operation:
# Search an element

```
#include <stdio.h>
#define MAX_SIZE 100 // Maximum array size
int main()
{
    int arr[MAX_SIZE];
    int size, i, toSearch, found;
    printf("Enter size of array: ");
    scanf("%d", &size);
    printf("Enter elements in array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("\nEnter element to search: ");
    scanf("%d", &toSearch);
```

```
found = 0; If element does not exist in array */
    for(i=0; i<size; i++)
    {  if(arr[i] == toSearch)
        { found = 1;
            break;}}
if(found == 1)
    { printf("\n%d is found at position %d",
toSearch, i + 1);
    }
    else  { printf("\n%d is not found in the array",
toSearch); }
  return 0;}
```

# Update Array Element

- We can **update the value of an element** at the given index i in a similar way to accessing an element by using the array subscript operator [ ] and assignment operator =

**array_name[i] = new_value;**

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};  // Initialize an array of 5 elements
    int i, new_value;

    // Ask for the index and new value
    printf("Enter the index to update (0-4): ");
    scanf("%d", &i);

    // Ensure valid index
    if (i >= 0 && i < 5) {
        printf("Enter the new value: ");
        scanf("%d", &new_value);

        // Update the value at the given index
        arr[i] = new_value;

    // Print the updated array
        printf("Updated array: ");
        for (int j = 0; j < 5; j++) {
            printf("%d ", arr[j]);
        }
    } else {
        printf("Invalid index!\n");
    }

    return 0;
}
```

# Example: Write a C program that calculates the average of different ages

```
int ages[] = {20, 22, 18, 35, 48, 26, 87, 70}; // An array storing different ages
float avg, sum = 0;
int i;
int length = sizeof(ages) / sizeof(ages[0]); // Get the length of the array
for (i = 0; i < length; i++) {  // Loop through the elements of the array
  sum += ages[i];
}
avg = sum / length; // Calculate the average by dividing the sum by the length
printf("The average age is: %.2f", avg); // Print the average
```

# Write a program that finds the smallest age among different ages

```
// An array storing different ages
int ages[] = {20, 22, 18, 35, 48, 26, 87, 70};
int i;
// Get the length of the array
int length = sizeof(ages) / sizeof(ages[0]);
// Create a variable and assign the first array
element of ages to it
int lowestAge = ages[0];
 // Loop through the elements of the ages array
to find the lowest age
for (i = 0; i < length; i++) {
  if (lowestAge > ages[i]) {
   lowestAge = ages[i];
  }}
```

LA A

~~6~~ 5

5

5 6

LA A

# Multidimensional Arrays – 2D and 3D

- A multi-dimensional array can be defined as an array that has more than one dimension.
- It can grow in multiple directions.
- **Syntax:**
- The general form of declaring N-dimensional arrays is shown below:

- type arr_name[size1][size2]….[sizeN];
- Ex.
  - ***Two-dimensional array:*** *int two_d[10][20];*
  - ***Three-dimensional array:*** *int three_d[10][20][30];*

# Size of Multidimensional Arrays

- The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of both dimensions.
- Example:
  - The array **arr[10][20]** can store total of (10*20) = 200 elements.
- To get the size in bytes, we multiply the size of a single element (in bytes) by the total number of elements in the array.
- Example:
  - The size of array **int arr[10][20] = 10 * 20 * 4 = 800 bytes,** where the size of int is 4 bytes.

# 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 |

# 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.

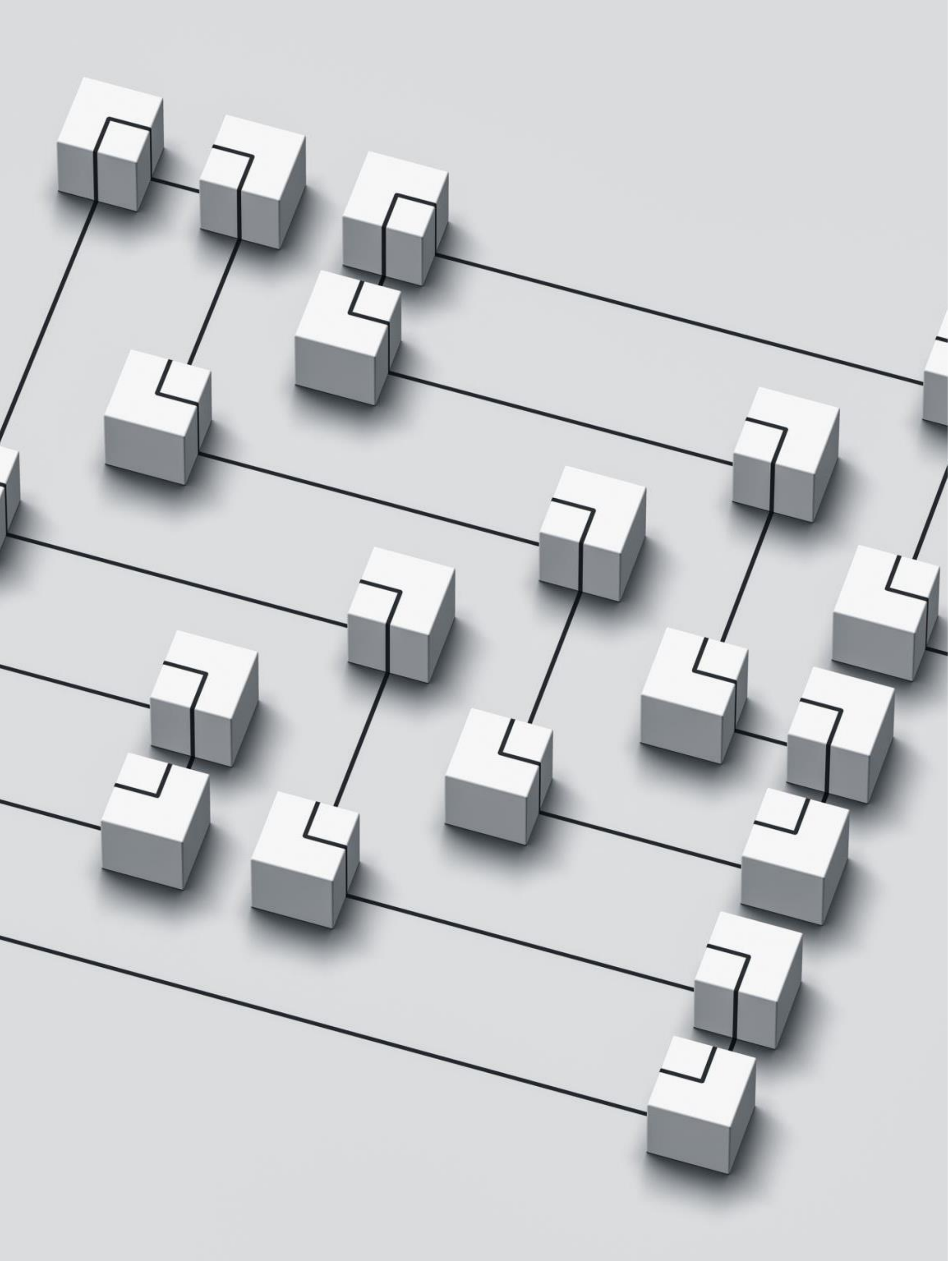
[1][2]

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

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

# Quiz 1 Discussion & Assignment Submission

- We will show the answer sheet today from 12:00 to 1:00 at #D313
- Assignment Submission on Blackboard is compulsory.



# Upcoming Slides

- Multi-dimensional arrays
- Functions