Lecture 05: Indexed Variables: Arrays

Course : Introduction to Programming

Dr. Farouk KECITA

Department of Process Engineering
Faculty of Science and Technology
University of Mila

Level: 1st year ST - ENG & LMD

Academic Year 2025/2026

Outline

1. Indexed Variables
2. Array Fundamentals
   - Definition and Terminologies
   - Characteristics
3. Array Declaration
4. Types of Arrays
   - One-Dimensional Arrays
5. Storing Values in Arrays
   - Three Methods
   - Initialization Methods
   - Assignment Method
   - Input Method
6. Accessing Array Elements
7. Arrays and Pointers
8. Summary

Why Arrays?

The Problem

So far we have used only single variable names with different types where each variable is associated with a single memory space for storing one data item.

Real Example

Storing marks of 200 students would require 200 separate variables: mark1, mark2, ..., mark200

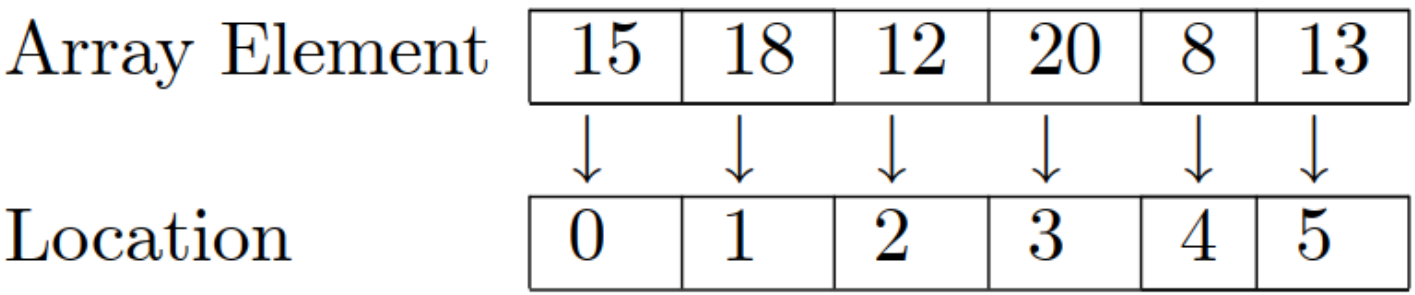
The Solution: Arrays

Arrays allow us to store multiple data items of the same type under one variable name in contiguous memory locations.

Definition of Arrays

Definition

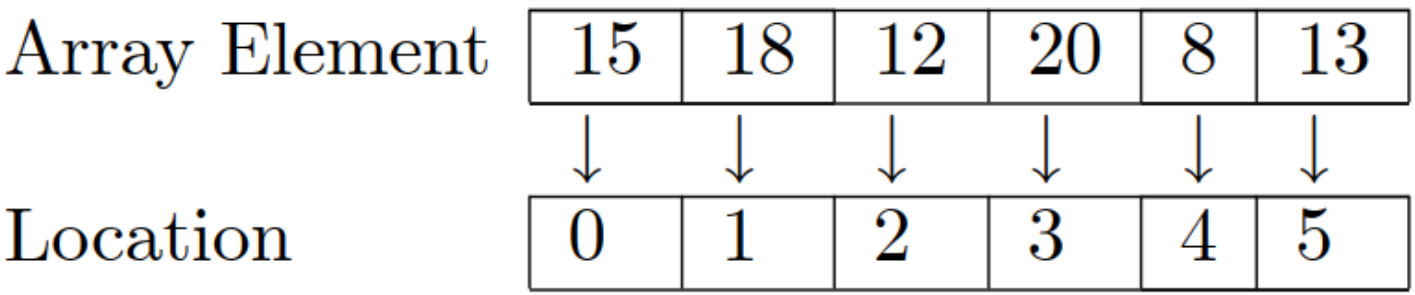
An array is a linear and homogeneous (similar) data structure that can store a fixed-size sequential collection of elements of the same type.



Definition of Arrays

Definition

An array is a linear and homogeneous (similar) data structure that can store a fixed-size sequential collection of elements of the same type.

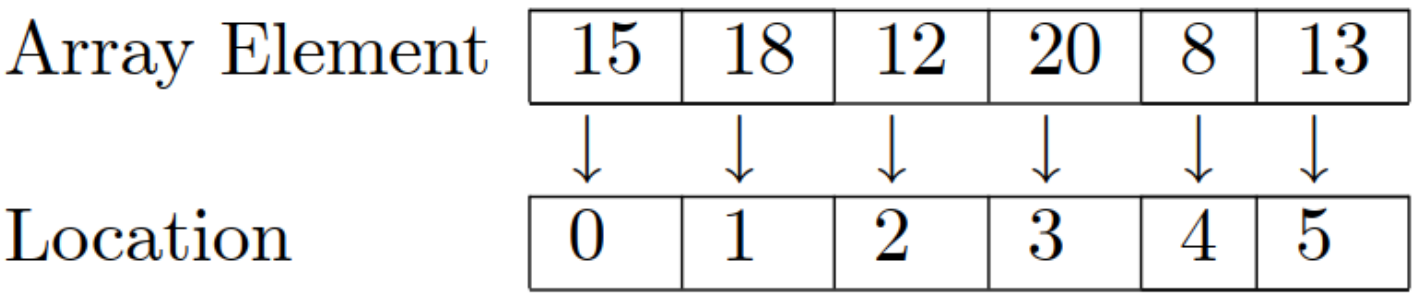


- Elements are stored **contiguously** in memory

Definition of Arrays

Definition

An array is a linear and homogeneous (similar) data structure that can store a fixed-size sequential collection of elements of the same type.

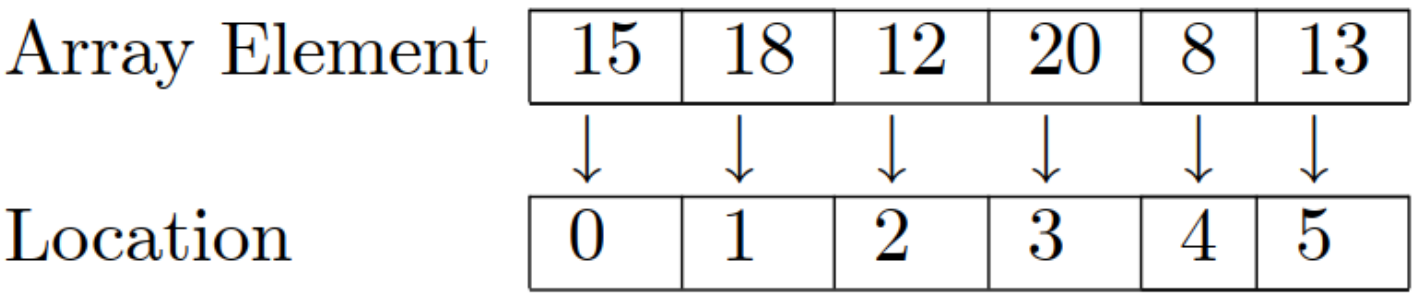


- Elements are stored **contiguously** in memory
- All elements share the same **variable name**

Definition of Arrays

Definition

An array is a linear and homogeneous (similar) data structure that can store a fixed-size sequential collection of elements of the same type.

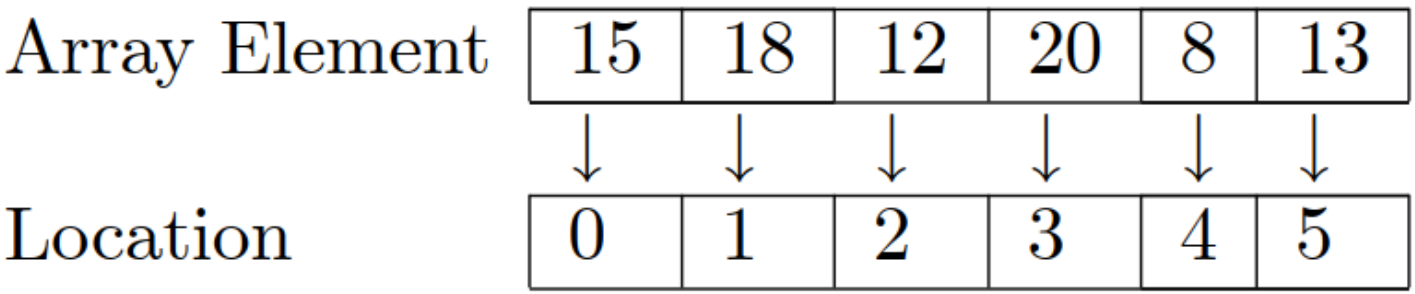


- Elements are stored **contiguously** in memory
- All elements share the same **variable name**
- Each element is accessed using an **index**

Definition of Arrays

Definition

An array is a linear and homogeneous (similar) data structure that can store a fixed-size sequential collection of elements of the same type.



- Elements are stored **contiguously** in memory
- All elements share the same **variable name**
- Each element is accessed using an **index**
- Index always starts from **0**

Array Terminologies

Size

Number of elements or capacity to store elements in an array.
Indicated in square brackets [].

Type

Data type of elements stored in the array. Determines memory allocation.

Index

Used to refer to array elements.
For example: `arr[i]`

Important

Index value is always an integer beginning from 0.

Characteristics of Arrays

- `int arr[5]` creates five integer-type variables in memory

Example

```
int arr[5] = {10, 20, 30, 40, 50};
arr[2] = 35;    // Changes only the third element
```

Characteristics of Arrays

- `int arr[5]` creates five integer-type variables in memory
- All elements share the same name, distinguished by index

Example

```
int arr[5] = {10, 20, 30, 40, 50};
arr[2] = 35;    // Changes only the third element
```

Characteristics of Arrays

- `int arr[5]` creates five integer-type variables in memory
- All elements share the same name, distinguished by index
- Elements can be modified individually without affecting others

Example

```
int arr[5] = {10, 20, 30, 40, 50};
arr[2] = 35;    // Changes only the third element
```

Characteristics of Arrays

- `int arr[5]` creates five integer-type variables in memory
- All elements share the same name, distinguished by index
- Elements can be modified individually without affecting others
- Array name represents the base address in memory

Example

```
int arr[5] = {10, 20, 30, 40, 50};
arr[2] = 35;    // Changes only the third element
```

Declaring Arrays

Syntax

```
<data type> arrayName[size1][size2]....[sizen];
```

Examples

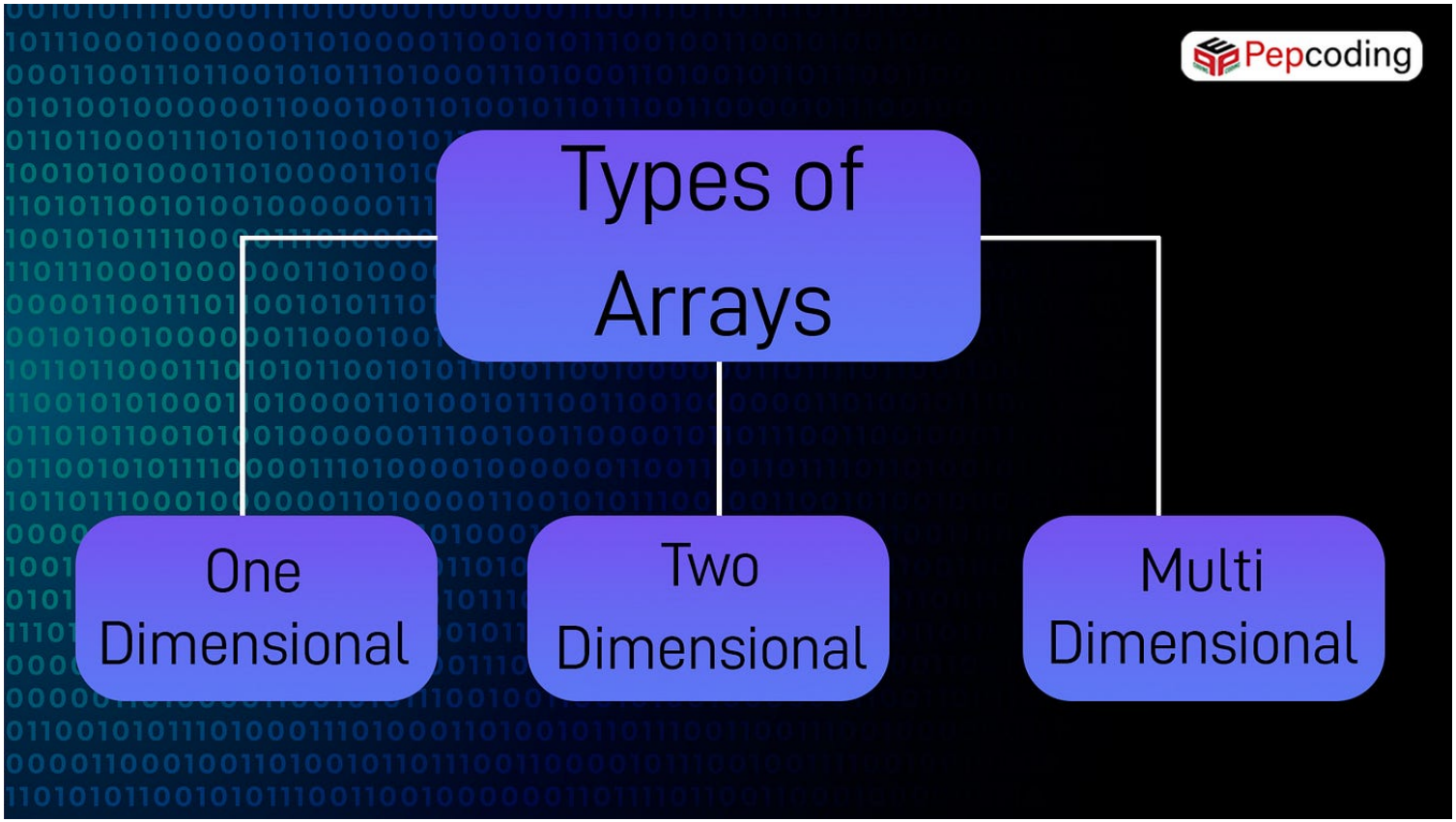
```
1 int arr_1[10];     // Array of 10 integers
2 char arr_2[15];    // Array of 15 characters
3 float arr_3[4];    // Array of 4 floating-point numbers
```

Compiler Information

The declaration tells the compiler:

1. Type and Name of the Array
2. Number of Array Dimensions
3. Number of Elements in Each Dimension

Types of Arrays



- **One-Dimensional Array (1D)**: Linear array with one index
- **Two-Dimensional Array (2D)**: Array with rows and columns
- **Multi-Dimensional Array**: Arrays with more than two dimensions

One-Dimensional Array (Linear Array)

Definition

Array having only one subscript variable (one index) is called One-Dimensional (1-D) array.

Declaration Syntax

```
<data type> <array name> [size];
```

```
1 int arr_1[10];     // 1D array of integers
2 char arr_2[15];    // 1D array of characters
3 float arr_3[4];    // 1D array of floats
```

Methods to Store Values in Arrays

1. **Initialization** - Assign values at declaration

Important

Declaration only allocates memory. Values must be stored explicitly.

Methods to Store Values in Arrays

1. **Initialization** - Assign values at declaration
2. **Assignment** - Assign values to individual elements

Important

Declaration only allocates memory. Values must be stored explicitly.

Methods to Store Values in Arrays

1. **Initialization** - Assign values at declaration
2. **Assignment** - Assign values to individual elements
3. **Input** - Read values from keyboard using scanf()

Important

Declaration only allocates memory. Values must be stored explicitly.

Initialization of Arrays

Syntax

```
<data type> <array name> [size] = {val1, val2, ..., valn};
```

Examples

```
1 int arr_1[10] = {2, 3, 4, 13, 1, 2, 12, 14, 20, 89};
2 char arr_2[15] = "welcome";
3 float arr_3[4] = {1.5, 3.5, 14.5, 20.2};
```

Full Array Initialization

Initializing All Elements

```
1 int arr[6] = {10, 11, 13, 39, 50};
```

$$\text{int arr}[6]=\{10,11,13,20,39,50\};$$

10	11	13	20	39	50

Partial Array Initialization

Initializing Some Elements

```
1 int arr[6] = {10, 11, 13};
```

$$\text{int arr}[6]=\{10,11,13\};$$

10	11	13	0	0	0

Note

Remaining elements are initialized to zero automatically by the compiler.

Initialization Without Size

Compiler Determines Size

```
1 int arr[] = {10, 11, 13, 20, 39, 50};
```

Result

Compiler counts the number of elements (6) and sets the array size accordingly.

Complete Program Example

```c
#include <stdio.h>
int main() {
    int vec[] = {10, 11, 13, 20, 39, 50};
    int i;

    for (i = 0; i < 6; i++) {
        printf("\n Array Element vec[%d] = %d", i, vec[i]);
    }
    return 0;
}
```

Output

```
Array Element vec[0] = 10
Array Element vec[1] = 11
Array Element vec[2] = 13
Array Element vec[3] = 20
Array Element vec[4] = 39
Array Element vec[5] = 50
```

String Initialization

Character Array (String)

```
1 char str[] = "Welcome";
```

Important

String length is 7, but array size is 8 because of NULL character (\0).

W	e	l	c	o	m	e	\0
str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]

Assigning Values to Arrays

Individual Element Assignment

```
1 int vect[4];
2 vect[0] = 10;
3 vect[1] = 11;
4 vect[2] = 30;
5 vect[3] = 40;
```

Reading Array Elements from Keyboard

Using scanf()

```
1 // Reading individual elements
2 scanf("%d", &arr[0]);
3 scanf("%d", &arr[1]);
4
5 // Reading in a loop
6 for (i = 0; i < n; i++) {
7     printf("Enter element %d: ", i);
8     scanf("%d", &arr[i]);
9 }
```

Accessing and Printing Array Elements

Using printf()

```
// Printing individual elements
printf("%d", arr[0]);

// Printing in a loop
for (i = 0; i < n; i++) {
    printf("\n arr[%d] = %d", i, arr[i]);
}
```

Complete Program: Read and Print Array

```c
#include <stdio.h>
int main() {
    int i, N, arr[10];
    printf("Enter number of elements: ");
    scanf("%d", &N);

    if (N < 10) {
        printf("Enter array elements:\n");
        for(i = 0; i < N; i++) {
            scanf("%d", &arr[i]);
        }

        printf("Array elements are:\n");
        for(i = 0; i < N; i++) {
            printf("arr[%d] = %d\n", i, arr[i]);
        }
    }
    else {
        printf("Error: Array size exceeded!\n");
    }
    return 0;
}
```

Program Output

Run 1 - Valid Input

```
Enter number of elements: 3
Enter array elements:
2
3
5
Array elements are:
arr[0] = 2
arr[1] = 3
arr[2] = 5
```

Run 2 - Invalid Input

```
Enter number of elements: 15
Error: Array size exceeded!
```

Relationship Between Arrays and Pointers

Key Concept

In C, arrays and pointers are closely related. The array name is a constant pointer to the first element.

Example Array

```
int vect[] = {10, 11, 13, 20, 39};
```

Accessing Array Elements Using Pointers

Expression	Description	Value
`vect`	Base address	2293296
`*vect`	First element	10
`*(vect+0)`	First element	10
`*(vect+1)`	Second element	11

Important Formula

`arr[i]` is equivalent to `*(arr + i)`

Demonstration Program

```c
#include <stdio.h>
#include <conio.h>

void main() {
    int vec[] = {10, 11, 13, 20, 39};
    int i;

    for(i = 0; i < 5; i++) {
        printf("%d %d %d %d %d %d\n",
               vec, &vec[i], vec[i],
               *(i+vec), *(vec+i), i[vec]);
    }

    getch();
}
```

Program Output Explained

Expression	Meaning
`vec`	Base address (constant)
`&vec[i]`	Address of element i
`vec[i]`	Value at index i (standard notation)
`*(i+vec)`	Value using pointer arithmetic
`*(vec+i)`	Value using pointer arithmetic
`i[vec]`	Alternative notation (works in C)

Key Insight

All expressions except the first two return the same value - the element at position i!

Key Points Summary

- Arrays store multiple elements of the same type in contiguous memory

Key Points Summary

- Arrays store multiple elements of the same type in contiguous memory
- Array indices start from 0

Key Points Summary

- Arrays store multiple elements of the same type in contiguous memory
- Array indices start from 0
- Three ways to store values: initialization, assignment, input

Key Points Summary

- Arrays store multiple elements of the same type in contiguous memory
- Array indices start from 0
- Three ways to store values: initialization, assignment, input
- Strings are character arrays terminated by NULL (\0)

Key Points Summary

- Arrays store multiple elements of the same type in contiguous memory
- Array indices start from 0
- Three ways to store values: initialization, assignment, input
- Strings are character arrays terminated by NULL (\0)
- Array name represents base address

Key Points Summary

- Arrays store multiple elements of the same type in contiguous memory
- Array indices start from 0
- Three ways to store values: initialization, assignment, input
- Strings are character arrays terminated by NULL (\0)
- Array name represents base address
- `arr[i]` is equivalent to `*(arr + i)`