UNIVERSITY OF MILA

Faculty of Science and Technology

Department of Process Engineering

Level: 1st year ST - ENG & LMD

Course : Introduction to Programming

Lecture 05:

Indexed Variables: Arrays

---

by:

Dr. Farouk KECITA

Academic Year 2025/2026

1 Indexed Variables

So far we have used only single variable names with different types (integer, real and character) where each variable is associated with a single memory space for storing one data item. If we need to store multiple copies of the same data such as data tables (static tables, vectors, matrices, lists, etc.) then it is very difficult for the user. To overcome this difficulty we use another data structure called: Indexed variables or arrays.

1.1 Definition of Arrays

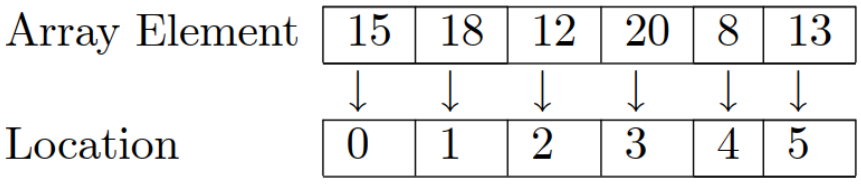
An array is a linear and homogeneous (similar) data structure that can store a fixed-size sequential collection of elements of the same type. It means that similar types of elements are stored contiguously in the memory under one variable name.

Key Points:

- Arrays store multiple elements of the same data type
- Elements are stored in contiguous memory locations
- All elements are accessed using an index
- Each index corresponds to one and only one element of the array

Example: Suppose we have to store marks of 200 students and print the largest one. Without arrays, we would have to declare 200 variables as mark1, mark2, mark3, ..., mark200 which is a very difficult task. Arrays allow us to store all 200 marks in contiguous memory under a single variable name.

1.2 Graphical Representation of C Programming Arrays



The above array is declared as: `int mark[6];`

```
mark[0] = 15; mark[1] = 18; mark[2] = 12; mark[3] = 20; mark[4] = 8; mark[5] = 13;
```

In the above figure, 15, 18, 12, 20, 8, 13 are actual data items. 0, 1, 2, 3, 4, 5 are index variables. To retrieve individual data elements, we use the name of the array and an integer enclosed in square brackets called subscript variable/index.

1.3 Array Terminologies

- Size: Number (quantity) of elements or capacity to store elements in an array. It is always indicated in square brackets [].
- Type: Refers to data type. It selects which type of element is stored in the array. It gives the compiler instructions on memory reservations based on data types.
- Index: The array name and square brackets are used to refer to the array element. For example arr[i], arr is the name of the array and i is index. The index value is always an integer value beginning from 0.

1.4 Characteristics of an Array

- The declaration `int arr[5]` simply creates five integer-type variables in memory instead of declaring five separate variables.
- All the elements of an array share the same name and they are distinguished from one another by the element number (index).
- The element index in an array plays a main role for accessing each element.
- Any particular element of an array arr[] can be modified separately, assigned, or equated to another ordinary variable or array variable of its type, without disturbing the other elements.

1.5 Declaring Arrays

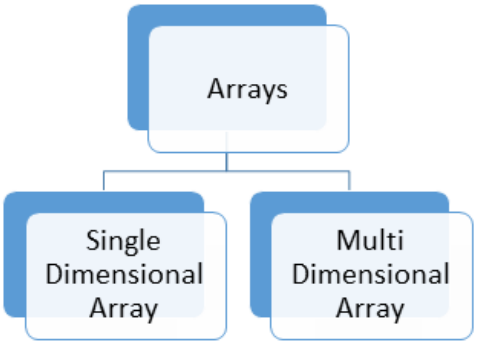
Arrays must be declared before using them in a C program. To declare an array, a programmer specifies the type of the elements and the number of elements required by an array as follows:

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

The declaration of an array tells the compiler that:

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

1.6 Types of Array



1.7 One Dimensional Array (Linear Array)

An array having only one subscript variable (one index) is called One-Dimensional (1-D) array or Single Dimensional Array. It is used to represent and store data in a linear form.

1.7.1 1-D Array Declaration

Syntax for declaration: `<data type> <array name> [size];`

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

Listing 1: 1-D Array Declaration Examples

1.7.2 Storing Values in Arrays

Declaration of arrays only allocates memory space for the array. But array elements are not initialized, hence values must be stored. There are three methods to store values in an array:

1. Initialization
2. Assigning Values
3. Input values from keyboard through scanf()

a) Initialization: Assigning the required values to array elements before processing is called initialization.

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

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

Listing 2: Array Initialization Examples

Different Methods of Initializing 1-D Array:

1) Initializing all elements of array: Arrays can be initialized at the time of declaration when their initial values are known in advance. In this method, all elements of the specified memory size are initialized.

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

10	11	13	20	39	50

2) Partial array initialization: If the number of values to be initialized is less than the size of the array, it is called partial array initialization. In such a case, elements are initialized in order from the 0^{th} element. The remaining elements will be initialized to zero automatically by the compiler.

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

10	11	13	0	0	0

3) Initialization without size: In this declaration, we do not provide a size to the array, but instead provide a set of values. The array size will be set to the total number of initial values specified.

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

Listing 3: Initialization Without Size

In the above example, the compiler counts the number of elements inside the braces and determines the size of the array (6 in this case).

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

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

```
    return 0;
}
```

Listing 4: Complete Program with Array Initialization

```
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
```

4) String Initialization: A string is a sequence of characters enclosed within double quotes (" ") but always ends with the NULL character (\0). The compiler puts (\0) at the end of the string to specify its termination.

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

Listing 5: String Initialization

Notice that the string length is 7, but the array size is 8 because we need one more location to store the 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]

b) Assigning values to arrays: Using assignment operators and indexing the array name, we can assign values to individual elements.

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

Listing 6: Assigning Values to Array Elements

c) Input values from keyboard through scanf(): To read values of a single dimensional array from the user (keyboard), we can use the scanf() function.

```
// To read 0th element:
scanf("%d", &arr[0]);

// To read 1st element:
scanf("%d", &arr[1]);

// In general, to read ith element:
```

```c
int arr[100], i, n;
printf("Enter number of elements: ");
scanf("%d", &n);

for (i = 0; i < n; i++) {
    printf("Enter array element %d: ", i);
    scanf("%d", &arr[i]);
}
```

Listing 7: Reading Array Elements from Keyboard

1.7.3 Accessing and Printing the 1-D Array

To print single dimensional array elements, we use the printf() function. Array elements are accessed using the array name and subscript variable inside square brackets.

```c
// To print 0th element:
printf("%d", arr[0]);

// In general, to print all elements:
int arr[100], i, n;
for (i = 0; i < n; i++) {
    printf("\n Array Element arr[%d] = %d", i, arr[i]);
}
```

Listing 8: Printing Array Elements

```c
#include <stdio.h>
int main() {
    int i, j, N, arr[10];

    printf("Enter number of array elements: ");
    scanf("%d", &N);

    if (N < 10) {
        // taking input and storing it in an array
        printf("Enter array elements:\n");
        for(i = 0; i < N; i++) {
            scanf("%d", &arr[i]);
        }

        // printing elements of an array
        printf("Array elements are:\n");
        for(j = 0; j < N; j++) {
            printf("arr[%d] = %d\n", j, arr[j]);
        }
    }
    else {
        printf("Error!! The number of elements that you want to
           enter in the ");
```

```
23          printf("array 'arr' is more than the array's size which
               is defined at compile time.\n");
24      }
25
26      return 0;
27  }
```

Listing 9: Complete Program: Read and Print Array Elements

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

Run 2:
Enter number of array elements: 15
Error!! The number of elements that you want to enter in the array 'arr'
is more than the array's size which is defined at compile time.
```

1.7.4 Accessing 1-D Array from Address

In C language, pointers and arrays are very closely related. If we have an initialized array, we can access the elements using a pointer. This method is very efficient compared to subscript notation.

To clarify further, consider the following example:

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

Expression	Description	Example (value)
`vect`	Returns the base address of the array vect	2293296
`*vect`	Gives zeroth element of the array vect	10
`*(vect+0)`	Gives zeroth element of the array vect	10
`*(vect+1)`	Gives first element of the array vect	11

Table 1: Accessing Array Elements Using Pointers

Whenever we access an array using `vec[i]`, it returns an element at the location `*(vec + i)`.

General formula: Accessing array `arr[i]` means retrieving element from address `(arr + i)`.

```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();
}
```

Listing 10: Program to Access Array Elements Using Different Methods

```
2293296 2293296 10 10 10 10
2293296 2293300 11 11 11 11
2293296 2293304 13 13 13 13
2293296 2293308 20 20 20 20
2293296 2293312 39 39 39 39
```

Note: The output shows that:

- `vec` (first column) always gives the base address
- `&vec[i]` (second column) gives the address of each element
- All other expressions (`vec[i]`, `*(i+vec)`, `*(vec+i)`, `i[vec]`) give the same value - the element at position i

Summary

- Arrays are homogeneous data structures storing elements of the same type in contiguous memory locations
- Array elements are accessed using indices starting from 0
- Arrays can be initialized at declaration, assigned values individually, or read from user input
- Arrays and pointers are closely related in C - `arr[i]` is equivalent to `*(arr + i)`
- Strings in C are character arrays terminated by the NULL character (\0)