Fundamentals of Programming

Lab Manual 7

Arrays in C Language

SCHOOL OF INTERDISCIPLINARY ENGINEERING AND SCIENCES NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY (NUST)

Objective:

The objective of this lab session is to understand the concept of arrays in the C programming language. Arrays are a fundamental data structure that allows storing multiple elements of the same data type under one name.

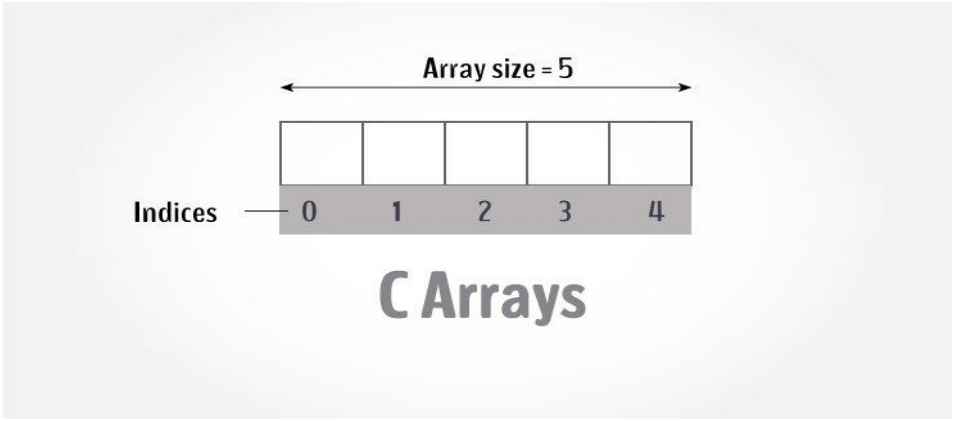
Arrays:

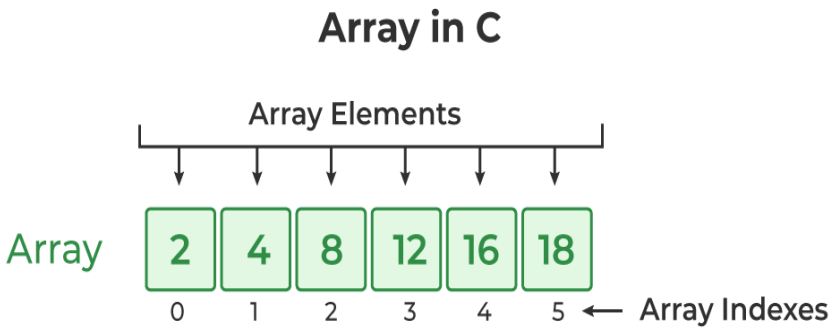
An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array.

```
int data[100];
```

An array is one of the most used data structures in C programming. It is a simple and fast way of storing multiple values under a single name.

An array in C is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and derived and user-defined data types such as pointers, structures, etc.





Array Declaration:

In C, we have to declare the array like any other variable before using it. We can declare an array by specifying its name, the type of its elements, and the size of its dimensions. When we declare an array in C, the compiler allocates the memory block of the specified size to the array name.

Syntax of Array Declaration:

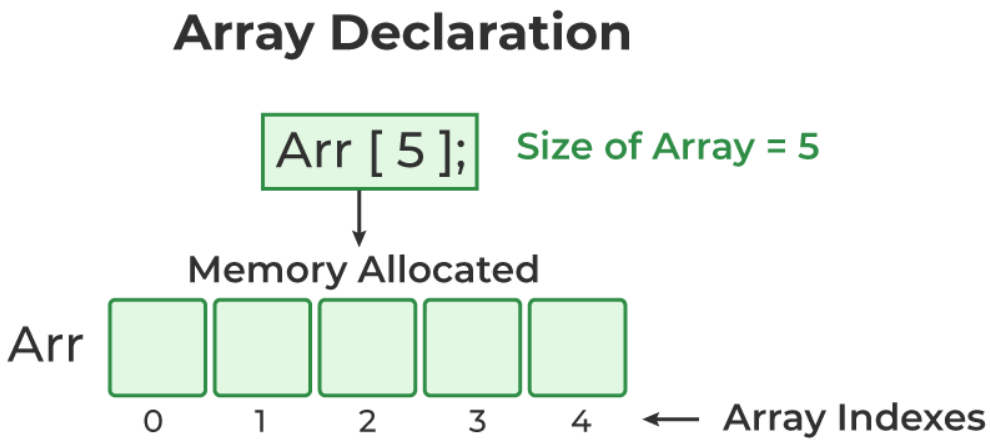
dataType arrayName[arraySize];

For example:

float mark[5];

Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

It's important to note that the size and type of an array cannot be changed once it is declared.



The C arrays are static, i.e., they are allocated memory at the compile time.

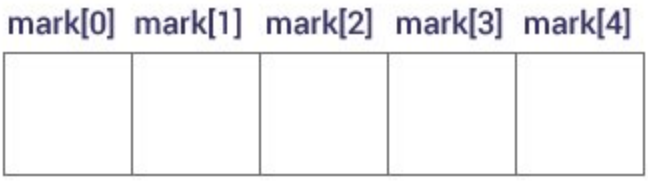
Code Example:

```
// C Program to illustrate the array declaration

#include <stdio.h>

int main()

{

        // declaring array of integers

        int arr_int[5];

        // declaring array of characters

        char arr_char[5];

        return 0;

}
```

Access Array Elements:

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



Few keynotes:

- Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4].

Code Example:

```
// C Program to illustrate element access using array
// subscript
#include <stdio.h>

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

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

        // accessing element at index 4 i.e last element
        printf("Element at arr[4]: %d\n", arr[4]);
        // accessing element at index 0 i.e first element
        printf("Element at arr[0]: %d", arr[0]);

        return 0;
}
```

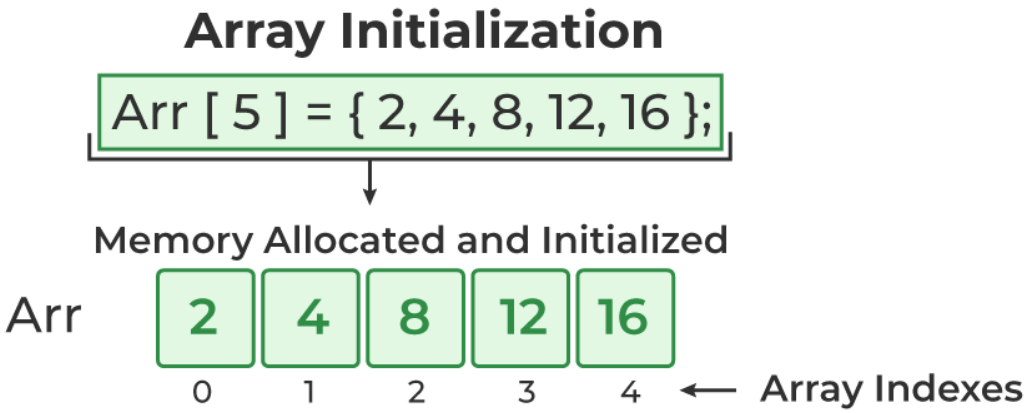
C Array Initialization:

Initialization in C is the process of assigning some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value. There are multiple ways in which we can initialize an array in C.

1. Array Initialization with Declaration:

In this method, we initialize the array along with its declaration. We use an initializer list to initialize multiple elements of the array. An initializer list is the list of values enclosed within braces **{ }** separated b a comma.

data_type array_name [size] = {value1, value2, ... valueN};



2. Array Initialization with Declaration without Size

If we initialize an array using an initializer list, we can skip declaring the size of the array as the compiler can automatically deduce the size of the array in these cases. The size of the array in these cases is equal to the number of elements present in the initializer list as the compiler can automatically deduce the size of the array.

data_type array_name[] = {1,2,3,4,5};

The size of the above arrays is 5 which is automatically deduced by the compiler.

3. 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 a for loop, while loop, or do-while loop to assign the value to each element of the array.

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

Code Example:

```
// C Program to demonstrate array initialization

#include <stdio.h>

int main()

{

        // array initialization using initialier list

        int arr[5] = { 10, 20, 30, 40, 50 };

        // array initialization using initializer list without specifying size

        int arr1[] = { 1, 2, 3, 4, 5 };

        // array initialization using for loop

        float arr2[5];

        for (int i = 0; i < 5; i++) {

                arr2[i] = (float)i * 2.1;

}

        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 =.

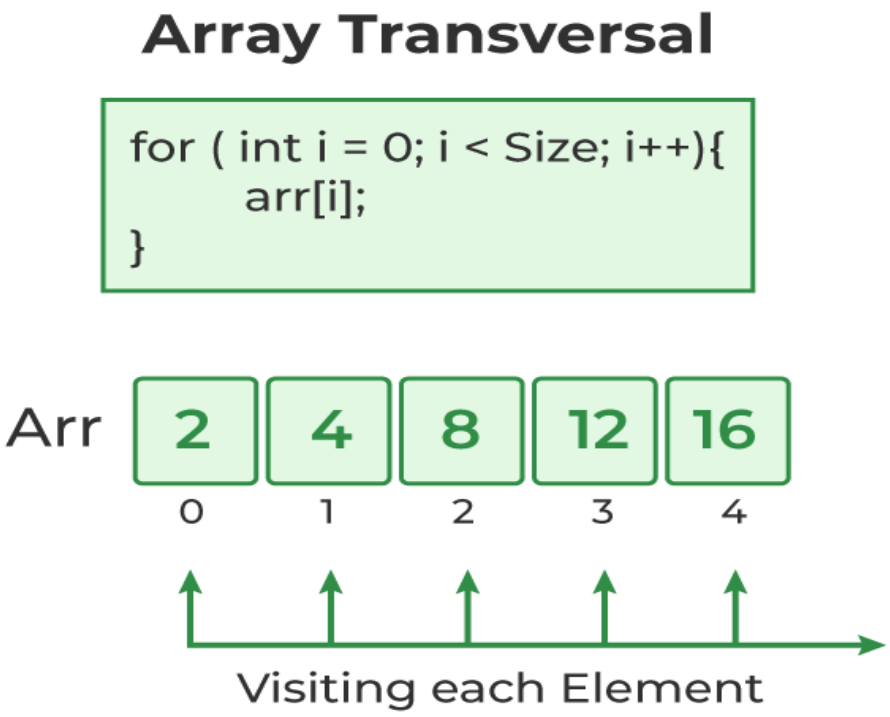
```
ARRAY_NAME[I] = NEW_VALUE;
```

Array Traversal:

Traversal is the process in which we visit every element of the data structure. For C array traversal, we use loops to iterate through each element of the array.

Array Traversal using for Loop:

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



Code Example:

```
// C Program to demonstrate the use of array
#include <stdio.h>

int main()
{
        // array declaration and initialization
        int arr[5] = { 10, 20, 30, 40, 50 };

        // modifying element at index 2
        arr[2] = 100;

        // traversing array using for loop
printf("Elements in Array: ");
        for (int i = 0; i < 5; i++) {
                printf("%d ", arr[i]);
        }

        return 0;
}
```

Types of Array in C

There are two types of arrays based on the number of dimensions it has. They are as follows:

1. One Dimensional Arrays (1D Array)
2. Multidimensional Arrays

1. One Dimensional Array in C:

The One-dimensional arrays, also known as 1-D arrays in C are those arrays that have only one dimension.

```
array_name [size];
```

Code Example:

```
// C Program to illustrate the use of 1D array
#include <stdio.h>
int main()
{
        // 1d array declaration
        int arr[5];
        // 1d array initialization using for loop
        for (int i = 0; i < 5; i++) {
                arr[i] = i ;
        }
        printf("Elements of Array: ");
        // printing 1d array by traversing using for loop
        for (int i = 0; i < 5; i++) {
                printf("%d ", arr[i]);
        }

        return 0;
}
```

Array of Characters (Strings):

In C, we store the words, i.e., a sequence of characters in the form of an array of characters terminated by a NULL character. These are called strings in the C language.

Code Example:

```
// C Program to illustrate strings
#include <stdio.h>
int main()
{
// creating array of character
char arr[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };
// printing string
int i = 0;
while (arr[i]) {
printf("%c", arr[i++]);
}
return 0;
}
```

Multidimensional Array in C:

Multi-dimensional Arrays in C are those arrays that have more than one dimension. Some of the popular multidimensional arrays are 2D arrays and 3D arrays. We can declare arrays with more dimensions than 3d arrays, but they are avoided as they get very complex and occupy a large amount of space.

1. **Two-Dimensional Array in C**

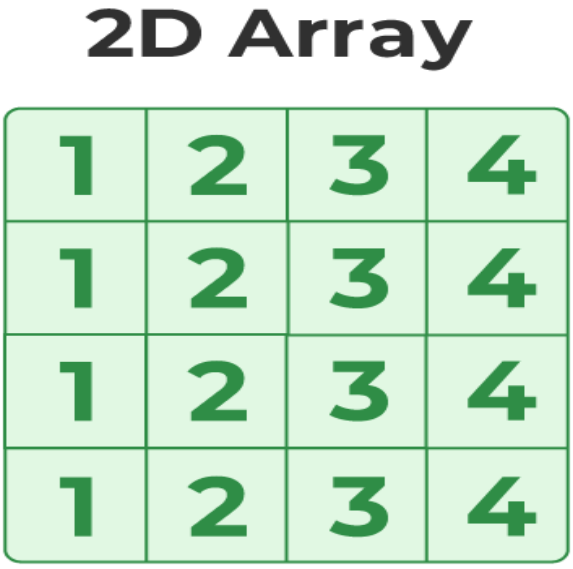
   A Two-Dimensional array or 2D array in C is an array that has exactly two dimensions. They can be visualized in the form of rows and columns organized in a two-dimensional plane.

   Syntax of 2D Array in C

   array_name[Size1] [Size2];

   Here,

   - Size 1: Size of the first dimension.
   - Size 2: Size of the second dimension.



Code Example:

```
// C Program to illustrate 2d array
#include <stdio.h>

int main()
{

        // declaring and initializing 2d array
        int arr[2][3] = {
                        {10, 20, 30},
                        {40, 50, 60}
                };
printf("2D Array:\n");
        // printing 2d array
        int i,j;
        for (i=0; i < 2; i++) {
                for (j=0; j < 3; j++) {
                        printf("%d ",arr[i][j]);
                }
                printf("\n");
        }

        return 0;
}
```

Output:

2D Array:

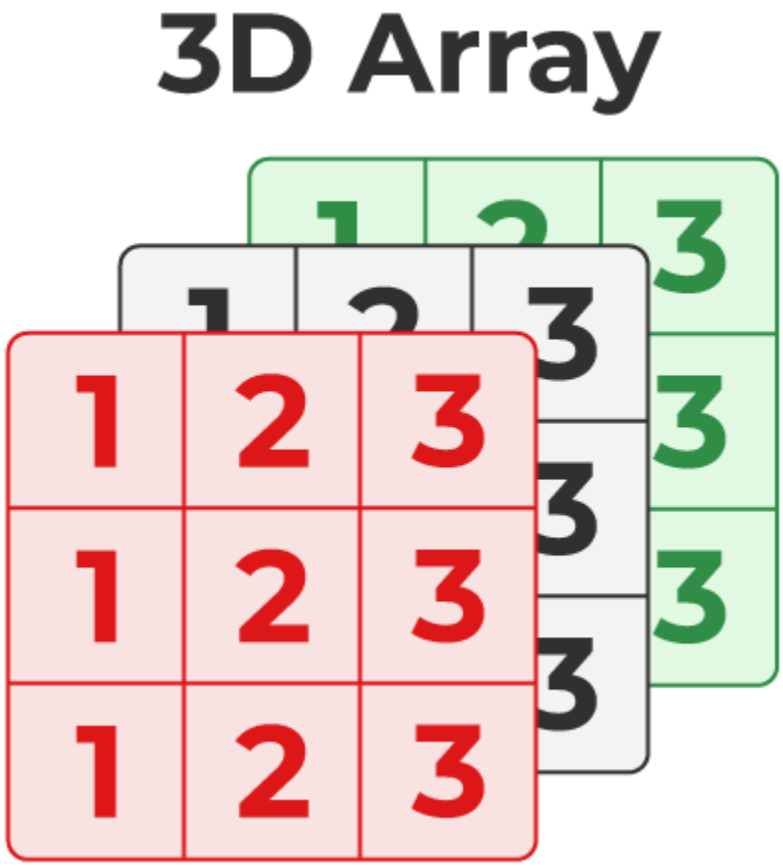
```
10 20 30
40 50 60
```

2. **Three-Dimensional Array in C**

Another popular form of a multi-dimensional array is Three-Dimensional Array or 3D Array. A 3D array has exactly three dimensions. It can be visualized as a collection of 2D arrays stacked on top of each other to create the third dimension.

Syntax of 3D Array in C

array_name [size1] [size2] [size3];



Code Example:

```
// C Program to illustrate the 3d array
#include <stdio.h>

int main()
{
```

```
        // 3D array declaration
        int arr[2][2][2] = {
                            {10, 20},
                            {30, 40},
                            {50, 60}
                             };

        // printing elements
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                for (int k = 0; k < 2; k++) {
                    printf("%d ", arr[i][j][k]);
                }
                printf("\n");
            }
            printf("\n \n");
        }
        return 0;
}
```

OUTPUT:

```
10 20
30 40


50 60
0 0
```

TASK 1:

Title: Finding the Maximum Element in an Array

Problem Statement:

You are required to write a C program that takes an input array of size 5 from the user. The program should display the entered array and then find the maximum number from this array.

Instructions:

1. The program should prompt the user to enter 5 integers to fill the array.
2. Display the entered array to the user.
3. Find the maximum number from the entered array.
4. Display the maximum number found.

Constraints:

- The array size must be fixed to 5.
- All array elements are integers.

Example:

Input:

Enter 5 integers:

10 25 7 14 36

Output:

Entered array: [10, 25, 7, 14, 36]

Maximum number: 36

TASK 2:

Title: Vowel or Consonant Checker

Problem Statement:

Write a C program that checks whether a given character is a vowel or a consonant.

Instructions:

1. Initialize an array containing all the vowels (a, e, i, o, u) in lowercase and uppercase.
2. Prompt the user to input a character.
3. Check if the entered character is present in the array of vowels. If it is, then display that it is a vowel. Otherwise, display that it is a consonant.
4. Ensure that your program is case-insensitive, meaning it should consider both lowercase and uppercase characters.

Constraints:

- The user will input a single character.

Example:

INPUT:

ENTER A CHARACTER: A

OUTPUT:

'A' IS A VOWEL.

INPUT:

ENTER A CHARACTER: B

OUTPUT:

'B' IS A CONSONANT.

TASK 3:

Title: Finding the Index of a Number in an Array

Problem Statement: Write a C program that finds the index of a number input by the user in a given array.

Instructions:

1. Initialize an array of integers with predefined values.
2. Prompt the user to input a number.
3. Search for the input number in the array.
4. If the number is found, display its index (position) in the array. If it's not found, display a message indicating that the number is not present in the array.
5. Ensure that the array size is fixed and known at compile time.

Constraints:

- The array size must be fixed.
- All elements of the array are integers.
- Assume that the array does not contain duplicate values.

Example:

Given array: [10, 25, 7, 14, 36]

Input: 7

Output: Index of 7 in the array is 2

Given array: [10, 25, 7, 14, 36]

Input: 20

Output 20 is not present in the array.