

## Module 4

### Arrays

#### 4.1 Array Declaration & Initialization

If we deal with similar type of variables in more number at a time, we may have to write lengthy programs along with long list of variables. There should be more number of assignment statements in order to manipulate on variables. When the number of variables increases, the length of program also increase.

In the above situations described above, where more number of same types of variables is used, the concept of ARRAYS is employed in C language. These are very much helpful to store as well as retrieve the data of similar type.

An Array describes a contiguously allocated non-empty set of objects with the same data type. The using arrays many number of same type of variables can be grouped together. All the elements stored in the array will referred by a common name.

##### Definition of Array

Array can be defined as a collection of data objects which are stored in consecutive memory locations with a common variable name.

OR

Array can be defined as a group of values referred by the same variable name.

OR

An Array can be defined as a collection of data objects which are stored in consecutive memory locations with a common variable name.

The individual values present in the array are called elements of array. The array elements can be values or variables also.

##### Types of Array

###### 1. One Dimensional Array

An array with only one subscript is called as one-dimensional array or 1- d array. It is used to store a list of values, all of which share a common name and are separable by subscript values

###### 2. Two Dimensional Array

An array with two subscripts is termed as two-dimensional array. A two-dimensional array, it has a list of given variable -name using two subscripts. We know that a one-dimensional array can store a row of elements, so, a two-dimensional array enables us to store multiple rows of elements.

##### 1. One Dimensional Array

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

Array Length = 9

First Index = 0

Last Index = 8

##### Declaration of Array

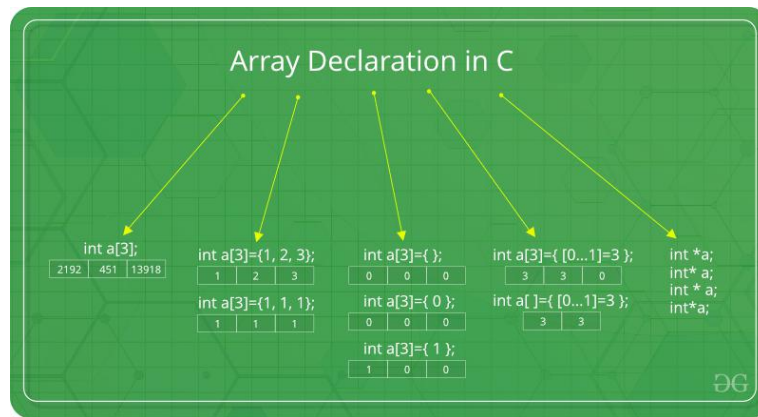
The general form for array declaration is

Type name[n] ; { one dimensional array}

Ex : int marks[20];

char name[15];

float values [ 10];



There are various ways in which we can declare an array. It can be done by specifying its type and size, by initializing it or both.

### 1. Array declaration by specifying size

```
// Array declaration by specifying size
int arr1[10];
```

### 2. Array declaration by initializing elements

```
// Array declaration by initializing elements
int arr[] = { 10, 20, 30, 40 }
```

```
// Compiler creates an array of size 4.
// above is same as "int arr[4] = {10, 20, 30, 40}"
```

### 3. Array declaration by specifying size and initializing elements

```
// Array declaration by specifying size and initializing
// elements
int arr[6] = { 10, 20, 30, 40 }
```

```
// Compiler creates an array of size 6, initializes first
// 4 elements as specified by user and rest two elements as 0.
// above is same as "int arr[] = {10, 20, 30, 40, 0, 0}"
```

### Array Initialization

Array can be made initialized at the time of declaration itself. The general form of array initialization is as below

```
type name[n] = [ element1, element2, .... element n];
```

The elements 1, element2... element n are the values of the elements in the array referenced by the same.

**Example1:-** `int codes[5] = [ 12,13,14,15,16];`

**Example2:-** `float a[3] = [ 1.2, 1.3, 1.4];`

**Example3:-** `char name [5] = [ 'S', 'U', 'N', 'I', 'L'];`

In above examples, let us consider one, it a character array with 5 elements and all the five elements are initialized to 5 different consecutive memory locations.

```
name[0] = 'S'
name[1] = 'U'
name[2] = 'N'
name[3] = 'I'
name[4] = 'L'
```

### Rules for Array Initialization

- Arrays are initialized with constants only.
- Arrays can be initialized without specifying the number of elements in square brackets and this number automatically obtained by the compiler.

- c) The middle elements of an array cannot be initialized. If we want to initialize any middle element then the initialization of previous elements is compulsory.
- d) If the array elements are not assigned explicitly, initial values will be set to zero automatically.
- e) If all the elements in the array are to be initialized with one and same value, then repetition of data is needed.

## 2. Two Dimensional Array

The basic form of declaring a two-dimensional array of size x, y:

**Syntax:**

```
data_type array_name[x][y];
```

data\_type: Type of data to be stored. Valid C/C++ data type.

We can declare a two dimensional integer array say 'x' of size 10,20 as:

```
int x[10][20];
```

Elements in two-dimensional arrays are commonly referred by  $x[i][j]$  where  $i$  is the row number and ' $j$ ' is the column number.

A two – dimensional array can be seen as a table with ' $x$ ' rows and ' $y$ ' columns where the row number ranges from 0 to  $(x-1)$  and column number ranges from 0 to  $(y-1)$ . A two – dimensional array 'x' with 3 rows and 3 columns is shown below:

	Column 0	Column 1	Column 2
Row 0	$x[0][0]$	$x[0][1]$	$x[0][2]$
Row 1	$x[1][0]$	$x[1][1]$	$x[1][2]$
Row 2	$x[2][0]$	$x[2][1]$	$x[2][2]$

**Initializing Two Dimensional Arrays:** There are two ways in which a Two-Dimensional array can be initialized.

**First Method:**

```
int x[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
```

The above array have 3 rows and 4 columns. The elements in the braces from left to right are stored in the table also from left to right. The elements will be filled in the array in the order, first 4 elements from the left in first row, next 4 elements in second row and so on.

**Better Method:**

```
int x[3][4] = {{0,1,2,3}, {4,5,6,7}, {8,9,10,11}};
```

This type of initialization make use of nested braces. Each set of inner braces represents one row. In the above example there are total three rows so there are three sets of inner braces.

**Accessing Elements of Two-Dimensional Arrays:** Elements in Two-Dimensional arrays are accessed using the row indexes and column indexes

$x[i][j]$  = element in  $i^{\text{th}}$  row and  $j^{\text{th}}$  column

## 4.2 Bound Checking

There is no index out of bounds checking in C, for example, the following program compiles fine but may produce unexpected output when run.

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

```
int main()
{
    int arr[2];

    printf("%d ", arr[3]);
    printf("%d ", arr[-2]);
}
```

```

    return 0;
}

```

In C, it is not compiler error to initialize an array with more elements than the specified size. For example, the below program compiles fine and shows just Warning.

```

#include <stdio.h>
int main()
{

    // Array declaration by initializing it with more
    // elements than specified size.
    int arr[2] = { 10, 20, 30, 40, 50 };

    return 0;
}

```

C don't provide any specification which deal with problem of accessing invalid index. As per ISO C standard it is called Undefined Behavior.

An undefined behavior (UB) is a result of executing computer code whose behavior is not prescribed by the language specification to which the code can adhere to, for the current state of the program (e.g. memory). This generally happens when the translator of the source code makes certain assumptions, but these assumptions are not satisfied during execution.

Examples of Undefined Behavior while accessing array out of bounds

1. **Access non allocated location of memory:** The program can access some piece of memory which is owned by it.

```

// Program to demonstrate
// accessing array out of bounds
#include <stdio.h>
int main()
{
    int arr[] = {1,2,3,4,5};
    printf("arr [0] is %d\n", arr[0]);

    // arr[10] is out of bound
    printf("arr[10] is %d\n", arr[10]);
    return 0;
}

```

2. **Segmentation fault:** The program can access some piece of memory which is not owned by it, which can cause crashing of program such as segmentation fault.

```

// Program to demonstrate
// accessing array out of bounds
#include <stdio.h>
int main()
{
    int arr[] = {1,2,3,4,5};
    printf("arr [0] is %d\n",arr[0]);
    printf("arr[10] is %d\n",arr[10]);

    // allocation memory to out of bound
    // element
    arr[10] = 11;
    printf("arr[10] is %d\n",arr[10]);
    return 0;
}

```

## 4.3 Character Arrays & Strings

**Declaration of Array of char type:** A string variable is any valid C variable name and is always declared as an array. The syntax of declaration of a string variable is:

```
char string-name[size];
```

The size determines the number of character in the string name.

**String:** Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

**Example:** An array of char type to store a string is to be declared as follows:

```
char str[8];
```

```
str[0] str[1] str[2] str[3] str[4] str[5] str[6] str[7]
```

P	r	o	g	r	a	m	\
---	---	---	---	---	---	---	---

An array of char is also called as a string variable, since it can store a string and it permits us to change its contents. In contrast, a sequence of characters enclosed within a pair of double quotes is called a string constant.

**Example:** "Program" is a string constant.

Initialization of Arrays of char Type

The syntax of initializing a string variable has two variations:

### Variation 1

```
char str1 [6] = { 'H', 'e', 'l', 'l', 'o', '\0'};
```

Here, str1 is declared to be a string variable with size six. It can store maximum six characters. The initializer – list consists of comma separated character constants. Note that the null character '\0' is clearly listed. This is required in this variation.

### Variation 2

```
char str2 [6] = { "Hello" };
```

Here, str2 is also declared to be a string variable of size six. It can store maximum six characters including null character. The initializer-list consists of a string constant. In this variation, null character '\0' will be automatically added to the end of string by the compiler.

In either of these variations, the size of the character array can be skipped, in which case, the size and the number of characters in the initializer-list would be automatically supplied by the compiler.

### Example

```
char str1 [ ] = { "Hello" };
```

The size of str1 would be six, five characters plus one for the null character '\0'.

```
char str2 [ ] = { 'H', 'e', 'l', 'l', 'o', '\0'};
```

The size of str2 would be six, five characters plus one for null character '\0'.

### String Handling Functions in 'C'

To perform manipulations on string data, there are built-in-function (library function) Supported by 'c' compiler. The string functions are.

1. STRLEN (S1)
2. STRCMP (S1, S2)
3. STRCPY (S1, S2)
4. STRCAT (S1, S2)

1. STRLEN (S1): This function is used to return the length of the string name S1,

Ex: S1 = 'MOID'

STRLEN (S1) = 4

2. STRCMP (S1, S2): This is a library function used to perform comparison between two strings. This function returns a value <zero when string S1 is less than S2. The function return a value 0 when S1=S2. Finally the function return a value > 0 when S1>S2.

3. STRCPY (S1, S2): This is a library function used to copy the string S2 to S1.

4. STRCAT (S1, S2): This is a library function used to join two strings one after the other. This function concatenates string S2 at the end of string S1.

**Example:** C program to concatenate the given two strings and print new string.

```
#include<stdio.h>

#include<conio.h>

main ( )
{
char s1[10], s2[10], s3[10],
int i,j,k;
printf ("Enter the first string : \n"); scanf ("%s",s1);
printf ("Enter the second string : \n");
scanf ("%s",s2);
i = strlen(s1);
j = strlen(s2);
for (k=0,k<i,k++) s3[k] = s1[k];
for (k=0,k<j,k++) s3[i+k] = s2[k];
s3[i+j] = '\0';
printf (" The new string is \n".s3);
}
```