UNIT 4

Arrays in 'C'

Structure

Learning Objectives

- To understand the importance of Array
- Definition of array
- Declaration and Initializing of an Array
- Types of Arrays
- Examples of an Array

4.0 Introduction

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.

4.1 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.

4.2 Types of Arrays

Basically arrays can divide in to

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.

4.2.1 Initialization of Array

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 area 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

1. Arrays are initialized with constants only.
2. Arrays can be initialized without specifying the number of elements in square brackets and this number automatically obtained by the compiler.
3. 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.
4. If the array elements are not assigned explicitly, initial values will be set to zero automatically.
5. If all the elements in the array are to be initialized with one and same value, then repletion of data is needed.

4.2.2 Declaration of Array

The array must be declared as other variables, before its usage in a C program. The array declaration included providing of the following information to C compiler.

- The type of the array (ex. int, float or char type)
- The name of the array (ex A[ ],B[ ] , etc)
- Number of subscripts in the array (i.e whether one – dimensional or Two-dimensional)
- Total number of memory locations to be allocated.

The name of the array can be kept by the user (the rule similar to naming to variable).

There is no limit one on dimensions as well as number of memory locations and it depends o the capacity of computer main memory..

The general form for array declaration is

Type name[n] ; { one dimensional array}

Ex : int marks[20];

char name[15];

float values [10];

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.

Declaration of One-dimensional Arrays:

The general form of declaring a one-dimensional array is

data-type	array-name [size];

Where data-type refers to any data type supported by C, array-name should be a valid C identifier; the size indicates the maximum number of storage locations (elements) that can be stored.

Each element in the array is referenced by the array name followed by a pair of square brackets enclosing a subscript value. The subscript value is indexed

from 0 to size -1. When the subscript vale is 0, first element in the array is selected, when the subscript value is 1, second element is selected and so on.

Example

int x [6];

Here, x is declared to be an array of int type and of size six. Six contiguous memory locations get allocated as shown below to store six integer values.

1	2	3	4	5	6
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]

Each data item in the array x is identified by the array name x followed by a pair of square brackets enclosing a subscript value. The subscript value is indexed from 0 to 5. i.e., x[0] denotes first data item, x[1] denotes second data item and x[5] denotes the last data item.

Initialization of One-Dimensional Arrays:

Just as we initialize ordinary variables, we can initialize one-dimensional arrays also, i.e., locations of the arrays can be given values while they are declared.

The general form of initializing an array of one-dimension is as follows:

data - type array - name [size] = {list of values};

The values in the list are separated by commas.

Example

int x [6] = {1, 2, 3, 4, 5, 6 };

as a result of this, memory locations of x get filled up as follows:

1	2	3	4	5	6
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]

Points to be considered during the declaration

1. If the number of values in initialization value - is less then the size of an array, only those many first locations of the array are assigned the values. The remaining locations are assigned zero.

Example: int x [6] = {7, 8, 6 };

The size of the array x is six, initialization value - consists of only three locations get 0 assigned to them automatically, as follows:

7	8	6	0	0	0
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]

2. If the number of values listed within initialization value - list for any array is greater than the six the array, compiler raises an error.

Example:

int x [6] = {1, 2, 3, 4, 5, 6, 7, 8 };

The size of the array x is six. But the number of values listed within the initialization - list is eight. This is illegal.

3. If a static array is declared without initialization value - list then the all locations are set to zero.

Example:

static int x [6];

0	0	0	0	0	0
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]

4. If size is omitted in a 1-d array declaration, which is initialized, the compiler will supply this value by examining the number of values in the initialization value - list.

Example:

int x [ ] = {1, 2, 3, 4, 5, 6 };

0	0	0	0	0	0
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]

Since the number of values in the initialization value-list for the array x is six, the size of x is automatically supplied as six.

5. There is no array bound checking mechanism built into C-compiler. It is the responsibility of the programmer to see to it that the subscript value does not go beyond size-1. If it does, the system may crash.

Example:

int x [6];

x [7] = 20;

Here, x [7] does not belong to the array x, it may belong to some other program (for example, operating system) writing into the location may lead to unpredictable results or even to system crash.

6. Array elements can not be initialized selectively.

Example:

An attempt to initialize only 2nd location is illegal, i.e.,

int x [6] = { , 10 } is illegal.

Similar to arrays of int type, we can even declare arrays of other data types supported by C, also.

Example

char ch [6];

ch[0]	ch[1]	ch[2]	ch[3]	ch[4]	ch[5]

ch is declared to be an array of char type and size 6 and it can accommodate 6 characters.

Example:

float x [6];

x is declared to be an array of float type and size 6 and it can accommodate 6 values of float type. Following is the scheme of memory allocation for the array x:

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]

Note

A one array is used to store a group of values. A loop (using, for loop) is used to access each value in the group.

Example

Program to illustrate declaration, initialization of a 1-d array

#include<stdio.h>

```
#include<conio.h>
void main( )
{
        int i, x [6] = {1, 2, 3, 4, 5, 6 };
        clrscr( );
        printf("The elements of array x \n");
        for(i=0; i<6; i++)
                printf("%d", x [i]);
        getch( );
}
```

Input – Output:

The elements of array x

1 2 3 4 5 6

4.3 Two - Dimenstional 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.

Example*:* Table of elements or a Matrix.

Syntax of two-dimensional arrays:

The syntax of declaring a two-dimensional array is:

```
data - type array - name [rowsize] [ colsize];
```

Where, data-type refers to any valid C data type, array -name refers to any valid C identifier, row size indicates the number of rows and column size indicates the number of elements in each column.

Row size and column size should be integer constants.

Total number of location allocated = (row size * column size).

Each element in a 2-d array is identified by the array name followed by a pair of square brackets enclosing its row-number, followed by a pair of square brackets enclosing its column-number.

Row-number range from 0 to row size-1 and column-number range from 0 to column size-1.

Example: int y [3] [3];

y is declared to be an array of two dimensions and of data type Integer (int), row size and column size of y are 3 and 3, respectively. Memory gets allocated to store the array y as follows. It is important to note that y is the common name shared by all the elements of the array.

Column numbers

Row numbers	0	1	2
1	x[0] [0]	x[0] [1]	x[0] [2]
2	x[1] [0]	x[1] [1]	x[1] [2]
3	x[2] [0]	x[2] [1]	x[2] [2]

Each data item in the array y is identifiable by specifying the array name y followed by a pair of square brackets enclosing row number, followed by a pair of square brackets enclosing column number.

Row-number ranges from 0 to 2. that is, first row is identified by row-number 0, second row is identified by row-number 1 and so on.

Similarly, column-number ranges from 0 to 2. First column is identified by column-number 0, second column is identified by column-number 1 and so on.

x [0] [0] refers to data item in the first row and first column

x [0] [2] refers to data item in the first row and third column

x [2] [3] refers to data item in the third row and fourth column

x [3] [4] refers to data item in the fourth row and fifth column

Initialization of Two-Dimensional Array

There are two forms of initializing a 2-d array.

First form of initializing a 2-d array is as follows:

data - type array - name [rowsize][colsize] = [initializer - list];

Where, data-name refers to any data type supported by C. Array-name refers to any valid C identifier. Row size indicates the number of rows, column size indicates the number of columns of the array. initialier-list is a comma separated list of values.

If the number of values in initializer-list is equal to the product of row size and column size, the first row size values in the initializer-list will be assigned to the first row, the second row size values will be assigned to the second row of the array and so on

Example*:* int x [2] [4] = {1, 2, 3, 4, 5, 6, 7, 8 };

Since column size is 4, the first 4 values of the initializer-list are assigned to the first row of x and the next 4 values are assigned to the second row of x as shown hereinafter

1 2 3 4

5 6 7 8

Note: If the number of values in the initializer-list is less than the product of rowsize and colsize, only the first few matching locations of the array would get values from the initializer-list row-wise. The trailing unmatched locations would get zeros.

Example: int x [2] [4] = {1, 2, 3, 4};

The first row of x gets filled with the values in the initializer-list. The second row gets filled with zeros.

1	2	3	4
0	0	0	0

The second form of initializing a 2-d array is as follows:

data-type array-name [rowsize] [colsize]

= { {initializer-list1},

{initializer-list2}, ……};The values in initializer-list 1 are assigned to the locations in the first row. The values in initializer-list2 are assigned to the locations in the second row and so on.

Example: int x [2] [4] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 } };

As a result of this, the array x gets filled up as follows:

1	2	3	4
5	6	7	8

Note

1. If the number of values specified in any initializer-list is less than colsize of x, only those may first locations in the corresponding row would get these values. The remaining locations in that row would get 0.

Example: int x [2] [4] = { { 1, 2, 3 }, { 4, 5, 6, 7 } };

Since the first initializer-list has only three values, x [0] [0] is set to 1, x [0] [1] is set to 2, x [0] [2] is set to 3 and the fourth location in the first row is automatically set to 0.

1	2	3	0
4	5	6	7

2. If the number of values specified in any initializer-list is more than colsize of x, compiler reports an error.

Example: int x [2] [4] = { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9 } };

colsize is 4, but the number of values listed in the first row is 5. This results in compilation error.

3. Array elements can not be initialized selectively.

4. It is the responsibility of the programmer to ensure that the array bounds do not exceed the rowsize and colsize of the array. If they exceed, unpredictable results may be produced and even the program can result in system crash sometimes.

5. A 2-d array is used to store a table of values (matrix).

Similar to 2-d arrays of int type, we can declare 2-arrays of any other data type supported by C, also.

Example: float x [4] [5];

x is declared to be 2-d array of float type with 4 rows and 5 columns
double y [4] [5];

y is declared to be 2-d arrays of double type with 4 rows and 5 columns

Note