## 5.0 Introduction

Arrays are useful to refer separate variables which are the same type. i.e. Homogeneous referred by a single name. But, we may have situations where there is a need for us to refer to different types of data (Heterogeneous) in order to derive meaningful information.

Let us consider the details of an employee of an organization. His details include employer's number (Integer type), employee's name (Character type) , basic pay ( Integer type) and total salary ( Float data type) . All these details seem to be of different data types and if we group them together, they will result in giving useful information about employee of the organization.

In above said situations, C provides a special data types called Structure, which is highly helpful to organize different types of variables under a single name.

## 5.1 Definition of Structure

A group of one or more variables of different data types organized together under a single name is called **Structure.**

Or

A collection of heterogeneous (dissimilar) types of data grouped together under a single name is called a **Structure**.

A structure can be defined to be a group of logically related data items, which may be of different types, stored in contiguous memory locations, sharing a common name, but distinguished by its members.

Hence a structure can be viewed as a heterogeneous user-defined data type. It can be used to create variables, which can be manipulated in the same way as variables of built-in data types. It helps better organization and management of data in a program.

When a structure is defines the entire group s referenced through the structure name. The individual components present in the structure are called structure members and those can be accessed and processed separately.

## 5.2 Structure Declaration

The declaration of a structure specifies the grouping of various data items into a single unit without assigning any resources to them. The syntax for declaring a structure in C is as follows:

struct Structure Name

```
{
        Data Type   member-1;
        Data Type   member-2;
          ....      ....
        DataType    member-n;
};
```

The structure declaration starts with the structure header, which consists of the keyword '**struct**' followed by a tag. The tag serves as a structure name, which can be used for creating structure variables. The individual members of the structure are enclosed between the curly braces and they can be of the similar or dissimilar data types. The data type of each variable is specified in the individual member declarations.

**Example:**

Let us consider an employee database consisting of employee number, name, and salary. A structure declaration to hold this information is shown below:

```
struct employee
{
        int eno;
        char name [80];
        float sal;
};
```

The data items enclosed between curly braces in the above structure declaration are called structure elements or structure members.

Employee is the name of the structure and is called structure tag. Note that, some members of employee structure are integer type and some are character array type.

The individual members of a structure can be variables of built – in data types (int, char, float etc.), pointers, arrays, or even other structures. All member names within a particular structure must be different. However, member names may be the same as those of variables declared outside the structure. The individual members cannot be initialized inside the structure declaration.

***Note***

Normally, structure declarations appear at the beginning of the program file, before any variables or functions are declared.

They may also appear before the main ( ), along with macro definitions, such as #define.

In such cases, the declaration is global and can be used by other functions as well.

## 5.2.1 Structure Variables

Similar to other types of variables, the structure data type variables can be declared using structure definition.

```
struct
{
    int    rollno;
    char  name[20];
    float   average;
    a, b;
}
```

In the above structure definition, a and b are said to be structure type variables. 'a' is a structure type variable containing rollno, name average as members, which are of different data types. Similarly 'b' is also a structure type variable with the same members of 'a '.

## 5.2.2 Structure Initialization

The members of the structure can be initialized like other variables. This can be done at the time of declaration.

**Example 1**

```
struct
{
    int     day;
    int     month;
    int     year;
```

```
    }
date = { 25,06,2012};
```

i.e

```
date. day = 25
date. month = 06
date. year = 2012
```

**Example 2**

```
struct address
{
   char   name [20];
   char   desgn [10];
   char   place [10];
} ;
```

i.e

```
struct address my-add = { 'Sree', 'AKM', 'RREDDY');
```

i.e

```
my-add . name = 'Sree'
my-add . desgn = AKM
my-add . place = RREDDY
```

As seen above, the initial values for structure members must be enclosed with in a pair of curly braces. The values to be assigned to members must be placed in the same order as they are specified in structure definition, separated by commas. If some of the members of the structure are not initialized, then the c compiler automatically assigns a value 'zero' to them.

**Accessing of Structure Members**

As seen earlier, the structure can be individually identified using the period operator ( . ). After identification, we can access them by means of assigning some values to them as well as obtaining the stored values in structure members. The following program illustrates the accessing of the structure members.

Example: Write a C program, using structure definition to accept the time and display it.

```
/ * Program to accept time and display it */
# include <stdio.h>
main( )
{
    struct
{
    int hour, min;
    float seconds;
 } time;
    printf ( "Enter  time in Hours, min and Seconds\n");
    scanf ( " %d %d %f", &time . hour, & time . min, & time . seconds);
    printf ( " The accepted time is  %d %d %f " , time . hour, time . min, time
. seconds ");
}
```

### 5.2.3 Nested Structures

The structure is going to certain number of elements /members of different data types. If the members of a structure are of structure data type, it can be termed as structure with structure or nested structure.

**Examaple**

```
struct
{
    int    rollno;
    char  name[20];
    float  avgmarks;
    struct
    {
```

```
    int  day, mon, year;
    } dob'
  } student;
```

In the above declaration, student is a variable of structure type consisting of the members namely rollno, name, avgmarks and the structure variable dob.

The dob structure is within another structure **student** and thus structure is nested. In this type of definitions, the elements of the require structure can be referenced by specifying appropriate qualifications to it, using the period operator ( . ) .

For example, **student. dob. day** refers to the element day of the inner structure dob.

## 5.3 Structures and Arrays

Array is group of identical stored in consecutive memory locations with a single / common variable name. This concept can be used in connection with the structure in the following ways**.**

a. Array of structures

b. structures containing arrays (or) arrays within a structure

c. Arrays of structures contain arrays.

### 5.3.1 Array of Structures

Student details in a class can be stored using structure data type and the student details of entire class can be seen as an array of structure.

**Examaple**

```
struct student
{
   int  rollno;
   int   year;
  int tmarks;
}
struct student class[40];.
```