# IV Unit Second Part

## STRUCTURES

Structure is a very useful derived data type supported in c that allows grouping one or more variables of different data types with a single name.

The general syntax of structure is given below:

```
Struct <tagname>
{
   datatype membername1;
   datatype membername2;
};
```

## Declaration and Initialization:

Structure can be declared and initialized either above the main function or within the main function. The process of defining, declaring and initializing of structure is illustrated below:

| Above main function | Within main function |
|---|---|
| struct student<br>{<br>char name[20];<br>int rollno;<br>}s1={“xyz”,123};<br><br>Or<br><br>struct student<br>{<br>char name[20];<br>int rollno;<br>}s1;<br>struct student s1={“xyz”,123};<br><br>void main()<br>{<br><br>}<br><br>**Explanation:**<br>In the above example student is the structure name and s1 is the structure variable declared and the values are initialized to the structure variable within flower braces | void main()<br>{<br>struct student<br>{<br>char name[20];<br>int rollno;<br>} s1={“xyz”,123};<br><br>}<br><br>Or<br>void main()<br>{<br>struct student<br>{<br>char name[20];<br>int rollno;<br>} s1;<br>struct student s1={“xyz”,123};<br>}<br>**Explanation:**<br>In the above example student is the structure Name and s1 is the structure variable declared and the values are initialized to the structure an variable within flower braces |

## Accessing members of the Structure:

The member’s of the structure can be accessed with a variable declared of that structure type and with dot operator(**.**)

The general form is shown below:

Structure variable name.member name

For eg.,

```
struct student
{
  char name[20];
  int  rollno;
}s1;
void main()
{
    printf("enter the name and rollno:");
    scanf("%s%s%d",s1.name,&s1.rollno);
    printf("name=%s\n",s1.name);
    printf("rollno=%d\n",s1.rollno);
}
```

**Explanation:**

In the above example a **structure by name student** is declared with **members name declared as character array and rollno declared of type as int.**

The scanf statement shows how data is read for the members of the structure through structure variable likewise the printf statement shows how to print the data of the members of the structure.

---

## Nested Structures(or)Structure within Structure:

Structure defined within another structure is said to be nested structures. Below are given examples of the different ways of making a structure nested.

```
struct employee
{
    int empid;
   char designation[15];
   struct salary
   {
       int basic;
       int da;
   }s1;
}emp;
```

```
struct date
{
      int day;
      int month;
      int year;
};
Struct employee
{
     int empno;
     char name[20];
      float basic;
      struct date joindate;
}emp1;
```

In case of nested structures the accessing of data for members is done **by accessing the outer structure variable followed by the dot operator and then the inner structure variable followed by the dot operator and then the member name.**

**Explanation:**

For example in the above structure to read data for the member basic. It is written as

Scanf("%d",&emp1.s1.basic);

Where emp1 is the outer structure variable and s1 is the inner structure variable and basic is the member name.

**Explanation:**

For example in the above structure to read data for the member day. It is written as

Scanf("%d",&emp1.joindate.day);

Where emp1 is the outer structure variable and joindate is the inner structure variable and day is the member name.

## Structures and Functions:

Structure variable can be passed as argument to function in two ways that is by value and by reference. Below we will see with example the procedure of passing structure variable as arguments by value and by reference.

```
# include<stdio.h>
#include<conio.h>
struct student              // Structure declaration
{
    int roll;
    char name[15];
};

struct student fun(struct student);  //Function Prototype

void main()
{
    struct student s={24,"xyz"},x;
    x=fun(s); /* function call  with structure variable passed as argument */
    printf("%d %s",x.roll,x.name);
}

struct student fun (struct student b)
{
    b.roll=99;
    strcpy(b.name,"abc");
    return b;
}
```

**Explanation:**

In the above example the structure variable is passed as argument in the function call and received in an **argument by name x** declared of the same structure in the called function and the procedure of reading data to the member of the structure is the same as what we have discussed above.

## Array of Structures

### Declaration

As we know that structure is a derived data type that can group one or more variables of different data types. If the same structure is required for more than once then we declare the variable as array of structures.
The declaration of array of structures is illustrated below:

```
struct student
{
  int rollno;
  int marks1;
};
struct student s1[3];
```

In the above example s1 is a variable declared as array of structures, where s1 is the variable name and the value 3 enclosed in square brackets gives the size of the array variable.

## Accessing the structure with array of structure variables

The following segment of code illustrates the process of accessing the structure with array of structure variables.

```
for(i=0;i<3;i++)
{
    scanf( "%d",&s[i].rollno);
    scanf("%d",&s[i].marks1);
 }
```

Here s1 is the structure variable declared as an array,rollno and marks1 are members of the structure and I is subscript variable used to vary the index of the array variable.

The members of the structures are accessed by giving **the structure variable name followed by subscript variable enclosed in square brackets followed by dot operator and then the member name.**

## Arrays within structures

**Declaration**

Structure is a derived data type that can group one or more variables of different data types with a single name, if one of these variables or all of these variables is declared as an array variable then it becomes as use of arrays within structures.

Example that illustrates the declaration of array within structure is given below:

```
struct student
{
  int rollno;
  int marks[3];
};

struct student s1;
```

in the above example variable marks is declared as array of type int.

**Accessing data for array declared within structure**

The following segment of code illustrates the process of accessing arrays within structure.

```
for(i=0;i<3;i++)
   scanf("%d",&s1.marks[i]);
```

here s1 is the structure variable, marks is the member of structure declared as array variable and I is subscript variable used to vary the index of the array variable.

The members of the structures declared as array are accessed by giving **the structure variable name followed by dot operator and then the member name with subscript variable enclosed in square brackets.**

## Structures and Unions

| Structures | Unions |
|---|---|
| Structures are derived data types that allows to group one or more variables of different data type with a single name. | Unions are derived data types that allows to group one or more variables of different data type with single name. |
| **Syntax:**<br>struct <tagname><br>{<br>Data type member name;<br>Data type member name;<br>}; | **Syntax:**<br>union <tagname><br>{<br>Data type member name;<br>Data type member name;<br>}; |
| Size allocated to the structure variable is the **sum of** | Size allocated to the union variable is the **maximum** |

| **the bytes allocated to the members of the structure.** | **number of bytes allocated to any member of the structure**. |
|---|---|
| **Example:**<br>struct college<br>{<br>int collegeid;<br>char collegaddress[20];<br>}c;<br><br>Here c is the structure variable declared which takes a size of 22 bytes i.e., 2 bytes for the member college id and 20 bytes for the member collegeaddres. | **Example:**<br>union type<br>{<br>int x;<br>char y;<br>float z;<br>}u;<br>Here u is the union variable declared which takes a size of 4 bytes i.e., 2 bytes for member x, 1 byte for member y and 4 bytes for member z and the maximum of these is 4 bytes which will be the size allocated to the union variable. |

## Typedef

The keyword **typedef** allows us to define an existing data type with other name. The general form of typedef is given below:

typedef datatype newname;

For e.g.,

typedef int integer;
typedef float real;

In the above example, int is redefined as integer and float redefined as real.

Below we will see a simple example of how typedef is used.

```
#include <stdio.h>
void main()
{
typedef int integer;
typedef float real

integer rollno;
real per_marks;

printf("enter value rollno and per_marks:");
scanf("%d %f",&rollno,&per_marks);
}
```

## Bitfields:

**Bitfields** is a concept supported in programming language C that allows the programmer to allocate **bits** of memory to variables instead of **bytes** of memory.

If a variable needs store a value in between 0 and 7 then no need to allocate bytes of memory only bits of memory is sufficient and this can be done by accompanying the variable with colon( **:** ) followed by non-negative integer value.

This concept can be applied only to members of a structure.

The general form of usage of bit fields is given below:

```
Struct <tagname>
{
        Data type membername:integer value;
        Data type membername:integer value;
};
```

Where integer value specifies the no. of bits to be allocated for that membername.

For e.g.,

```
#include <stdio.h>
#include <conio.h>
void main()
{
        struct student
        {
                int date:5;
                 int month:4;
                 int year:12;
         }s;
          s.date=10;
          s.month=6;
          s.year=2013;
          printf("%d%d%d",s.date,s.month,s.year);
}
```

**Explanation:**

In the above program, **structure is defined by name student** comprising of members **date, month, year** with each member assigned with bits of memory. For the members defined in the structure **only bits of memory is allocated rather than bytes of memory.**

## ***Enumerations***

- An *enumeration* is a list of constant integer values.

Ex:

enum colors {RED,WHITE,BLUE};

By default, the first name in an enumeration has value 0, the next 1, and so on. In this example, RED = 0, WHITE = 1, BLUE = 2. The output of the following statement is 2.

printf("%d",BLUE);

- We can specify explicit values to enumerators with the help of *equal to* (=) operator. The subsequent enumerators continue from the assigned value by incrementing 1.

Ex:

enum branchcode {CIVIL=1,EEE,MECH,ECE,CSE};

Here, EEE is 2, MECH is 3, ECE is 4 and CSE is 5.

- Names in an enumeration must be distinct. Values in the same enumeration need not be distinct.

Ex: enum names {ASHA=28,SWAPNA=21,JOHN=5, HARI=46};

```
#include<stdio.h>
#include<conio.h>
void main()
{
        enum colors {RED,WHITE,BLUE};
        printf("%d %d %d",RED,WHITE,BLUE);
}
```

**Output: 0 1 2**

**Structure Pointer**

The way we can have a pointer pointing to an int, or a pointer pointing to a char, similarly we can have a pointer pointing to a struct. Such pointers are known as 'structure pointers'.

Using Structure Pointers:

To access the values of the members of the structure using pointers we need to perform following things,
Let's take the following segment of code

```
#include<stdio.h>
#include<conio.h>
void main( )
struct student
{
        int rno;
        char name[10];
        float avg;
};
struct student s={101,"ABC",78.98};
struct student *ptr;
ptr=&s;
printf("\n%d %s %f", s.rno,s.name,s.avg);
printf("\n%d %s %f", ptr → rno, ptr → name, ptr → avg);
getch();
}
```

ptr is a pointer to structure, then the members of the structure can also be accessed using selection operator denoted by →(which is formed by minus sign and greater than symbol).

Structure Containing Pointers

We can also contain pointers as members of the structures. The pointer is pointing to a value of the structure member or some other variable. The below code illustrates pointers as members of structures.

```
#include<stdio.h>
#include<conio.h>
struct student
{
        char name [10];
        int rno;
        float *ptr;
};
void main( )
{
        struct student s1= {"ABC",405};
        float avg=34.5;
        s1.ptr=&avg; // initializing pointer
        printf ("%s %d %f",s1.name,s1.rno,*s1.ptr); // accessing the value of a variable
}
```