# 8. Structure and Union

A structure is a collection of variable under a single name. The variables may be of different type and each has a name which is used to select it form structure. The variables are called member of structure. A structure is a convenient way of grouping several pieces of related information together. A structure combine several pieces of related information together. A structure is new name of heterogeneous data type or user defined data type. It may use other structures arrays or pointers as some of its members in a single unit.

An array can be used only to represent a group of data items that must be same type. Such as int, float. However if we want to represents collection of data items of different type using same name. Array cannot do so. In this situation, structure is used, which is method for packing data of different items. For example name roll, fees marks are related information of student. To store information about a student we would require to store the name of student which is array of characters, roll of student which is int, fees of student is in float similarly marks of student may be float and character of students may be m or f. there attributes of students are grouped together into single entity, student. Here student is known as structure which organized different data items in more meaningful way.

## 8.1. Defining a Structure

**Syntax:**

```
struct structure_name
{
        data type variable;
        date type variable;
        ……;
};
```

Once structure name is declared as new data type then, variables of that type can be declared as strut student name variable.

**Example:**

```
struct student
{
        char name [];
        int roll;
        float marks;
};
```

Here, student is structure name and its members are name, roll marks. The student is new data type and variable of structure student can be declared as: **struct student st;**

Here structure st is variable of type student. The multiple variables are declared as std2, std3, std3.

Each variables of structure has its own copy of member variables. The member variables are accessed using dot (.) operator. For example std1.name std1.roll, std1.marks.

# 8.2. Structure Initialization

Like standard data type, the members of a structure variable can be initialized. The values to be initialized must appear in order as in the definition of structure within brace and separated by comma. C does not allow initialization of individual structure member within its definition.

**Syntax:**

```
struct structure_name= {variable, variable, variable, variable};
```

**Example:**

```
struct student
{
        char name [];
        int roll;
        float marks ;
};
```

The variable of this type can be declared as: **strut student st= {"Yagya", 100, 34.54};** or is equivalent to **strut student st; st.name="Ram", st. Marks=34.54;st.roll=45;**

# 8.3. Processing and Accessing Member of Structure

The member of structure is usually processed individually, as separate entity. Therefore we must be able to access individual structure members. A structure member can be accessed by using period (.) operator.

**Syntax:**

**structure variable. member**

Example:

**std.name,std.id,std.marks**

## 8.3.1. Precedence of Dot Operator (.)

The dot operator is member of the highest precedence group and associates form left to right. Since it is an operator of highest precedence, dot operator will take first precede over various arithmetic, relational and logical assignment and unary operator. Thus ++st .marks is equivalent to ++(st. marks), implying that the dot operators acts first and then unary operator.

## 8.3.2. Structure Elements

The elements of structure are always stored in contiguous memory locations. A structure variable reverses number of bytes equal to sum of bytes needs to each of its members. For example int following structure students variable st takes 7 bytes in members are its member's variable( roll, 2 bytes marks needs 4 bytes and remarks need 1 byte.)

# 8.3. Array of Structure

Like array of int, float or char type there may be array of structure. In this case, the array will have individual structure as its elements. In our previous structure example. If we want to keep record of 50 students, we have to make 50 structure variables like std1, std2. But this technique is not good. At this situation we can use array of structure to store records of 50 students. Similarly to declaring structure variable an array of structure can be declared in two ways:

```
struct employee
{
        char name [];
        int id;
        float salary ;
} emp [10];

struct employee emp[10];
```

## 8.3.1. Initialization Array of Structure

We can initialize array of structure in same way as a single structure. This is illustrated as:

```
struct book
{
        char name [30];
        int page;
        float price;
} ;
struct book b[5]={"",45,45.3,"",30,50.90,"",45,80.89,"",230,809.89,"",67,68.34};
```

struct book[5] dimension means it reserve five location for each name, int page and float price.

# 8.4. Structure within Another Structure/Nested Structure

One structure can be nested within another structure in C. In other words the individual members of structure can be nested other on structure as well. Let us consider structure date which has member day, month and year. The date structure can be nested within another structure say person. Structure date is member of another structure person. This can be done as follows.

```
struct employee
{
        char name[];
        int id;
        float salary;
        struct
        {
                int day;
                int month;
                int year
        }date;
}emp[10];
```

This can be nested within another structure as its member.
Here, structure person contain member date which is itself a structure with there members. The members within structure date are accessed as  big structure variable as p.date.day, p.date.month.

# 8.5. Pointer to Structure

Pointer can be used with structure to store address of structure type variable; we can define a structure type pointer variable as normal way. Let us consider a structure book that has members name, page and price. It can be declared as:

```
struct book
{
        char name[45];
        int page;
        float price;
};
struct book b;  // this is normal variable;
struct book *bp; // here bp is pointer variable of structure book.
```

To use structure members through pointer memory must be allocated for a structure by using function call malloc() or by adding declaration and assignment as given bp=&b;

An individual structure member can be accessed in terms of its corresponding pointer variable by writing variable -> member. Here -> is called arrow operator and there must be structure on left side of this operator.

```
b.name or bp->name or (*bp).name
b.page or bp->page or (*bp).page
b.price or bp->price or (*bp).price
```

## 8.5.1. Function and Structure

Structure variable can also be passed to a function we may either pass individual structure elements or the entire structure.

## 8.5.2. Passing Structure Members to Functions

A structure member can be treated just as normal variable of  structure type. For example, integer structure variable can be treated just as integer. Thus structure members can be passed to function like ordinary variables.

## 8.5.3. Passing Whole Structure to Functions

It is possible to send entire structures functions as arguments in the functions call structure variable is treated as any ordinary variable.

## 8.5.4. Passing Structure Pointer to Functions

In this case the structure pointer is passed to function. If the pointer to a structure is passed as an argument to a function, then any changes that are made in the function are visible in the caller.

### 8.5.5. Passing Array of Structure to Function

Passing an array of structure to function involves same syntax and properties as passing array to a function. The passing is done using a pointer array structure in function if, any changes made in function to structure to the structure also visible in caller.

## 8.6. Union

Union are similar to structure its syntax and use. It also contains members whose individual data types may differ from one another (heterogeneous data type).

The distinction is that all members within union share the same storage area of computer memory where as each member within a structure is assigned its own unique storage. Thus unions are used to save memory space.

Since same memory is shared by all members. One variable can reside into another memory at a time. When another variable is set into memory of previous involving multiple members whose value need to be assigned to all members at the same time.

Therefore although union may contain many members of different types it can handled only one big member at a time. The compiler allocates a piece of storage that is large enough to hold largest variable type in union.

```
union student
{
        int roll;
        float marks ;
};
```

here union student allocates float data type within it int resides on.