variable and use it. But in the case of a structure, first we have to design and declare a data structure before the variables of that type are declared and uses.

Declaring Structure Variables :- It includes the following elements :-

1. The keyword Struct
2. The structure tag name
3. List of variable names separates by commas
4. A terminating semicolon

Accessing Structure Members :- The link bet" a member and a variable is established using the member operator '.' which is also known as 'dot operator' or 'period operator'.

Structure initialization :- like any other data, a structure variable can be initialized at compile time.

Rules for Initializing Structures :-

There are few rules to keep in mind while initializing structure variables at

Structures and Unions

Introduction:- A structure is a convenient tool for handling a group of logically related data items. Structure is a mechanism for packing data of different types. The keyword **struct** declares a structure. The general format of a structure definition is as follows:

```
struct tag-name
{
    data-type member1;
    data-type member2;
    :    :    :    :
};
```

Arrays Vs Structures

*) An array is a collection of related data elements of same type. Structure can have elements of different types.

*) An array is derived data type whereas a structure is a programmer defines one.

*) Any array behaves like a built in data type. All we have to do is to declare

compile-time.

* We cannot initialize individual members inside the structure template.
* The orders of values enclosed in braces must match the order of members in the structure definition.
* It is permitted to have a partial initialization. We can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.
* The uninitialized members will be assigned default values as follows:
  - ⇒ zero for integer and floating point numbers.
  - ⇒ '\0' for characters and strings.

Copying And Comparing Structure Variables :-

⇒ Two variables of the same structure type can be copied the same way as ordinary variables ex ⇒ student1 = student2;

Note :- C does not permit any logical operations on structure variables. In case, we need to compare them, we may do so by comparing members individually. ex:- student1 == student2 —x

in C. for example:-

```
struct student
  { char name[20];
    int rollno;
    struct
      { int sub[3];
        int total;
        int percentage;
      }marks;
  }stud1;
```

Structures And Functions:- There are three methods by which the values of a structure can be transferred from one function to another.

* The first method is to pass each member of the structure as an actual argument of the function call. The actual arguments are then treated independently like ordinary variables.

* The second method involves passing of a copy of the entire structure to the called function. Since the function is working on a copy of the structure any changes to structure members with in the function are not reflected in the original

structure. It is necessary for the function to return the entire structure back to the calling function.

*) The third method involves the concepts called pointers to pass the structure as an argument. In this case the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it.

x - x - x - x - x - x - x - x

```
struct
{ int count;
  float *p;      /* pointer inside the struct */
} ptr;           /* struct type pointer */
```

then the statement

++ptr -> count; increaments count not ptr.

=> (++ptr) -> count; increaments ptr first then link to count.

=> ptr++ -> count; increament ptr after accessing count.

1) *ptr -> p fetches whatever p points to.

2) *ptr -> p++ Increments p after accessing whatever it points to.

3) (*ptr -> p)++ Increments whatever p points to.

4) *ptr++ -> p Increments ptr after accessing whatever it points to.