Structure:

Why use structure?

In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity Student may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

- Construct individual arrays for storing names, roll numbers, and marks.
- Use a special data structure to store the collection of different data types.

What is Structure?

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information

The ,struct keyword is used to define the structure.

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```

Example to define a structure for an entity employee.

```
struct employee
{   int id;
    char name[20];
    float salary;
};
```

Here, struct is the keyword; employee is the name of the structure; id, name, and salary are the members or fields of the structure.

Declaring structure variable:

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1st way:

Example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{   int id;
    char name[50];
    float salary;
};
```

Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2nd way:

Another way to declare variable at the time of defining the structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;
```

Which approach is good?

If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.

If no. of variables are fixed, use 2nd approach. It saves your code to declare a variable in main() function.

Accessing members of the structure:

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

The code to access the id member of p1 variable by. (member) operator.

```
p1.id
```

Structure example:

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
}e1;  //declaring e1 variable for structure
int main( )
{
  //store first employee information
  e1.id=101;
  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```

Output:

```
employee 1 id : 101
employee 1 name : Sonoo Jaiswal
```

Union:

Like structure, Union in c language is *a user-defined data type* that is used to store the different type of elements.

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.

Advantage of union over structure:

It occupies less memory because it occupies the size of the largest member only.

Disadvantage of union over structure:

Only the last entered data can be stored in the union. It overwrites the data previously stored in the union.

Defining union:

The union keyword is used to define the union

Syntax to define union:

```
union union_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```

Example of union:

```
union employee
{   int id;
    char name[50];
}e1;  //declaring e1 variable for union
int main( )
{
   //store first employee information
   e1.id=101;
   strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
   return 0;
}
```

Output:

```
employee 1 id : 1869508435
employee 1 name : Sonoo Jaiswal
```

As you can see, id gets garbage value because name has large memory size. So only name will have actual value.