UNIT IV STRUCTURES AND UNION 9

Structure - Nested structures – Pointer and Structures – Array of structures – Self referential structures – Dynamic memory allocation - Singly linked list – typedef – Union - Storage classes and Visibility.

STRUCTURE

Structure in c language is a user defined data type that allows you to hold different type of elements. Each element of a structure is called a member. It works like a template in C++ and class in Java. You can have different type of elements in it. It is widely used to store student information, employee information, product information, book information etc.

Defining structure

The struct keyword is used to define structure. Let's see the syntax to define structure in c.

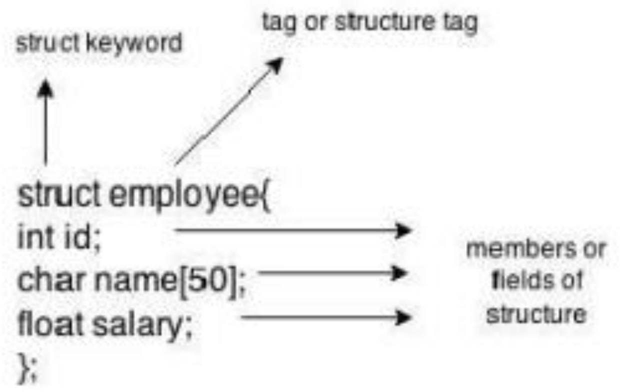
Let's see the syntax to define structure in c.

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

Let's see the example to define structure for employee in c.

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

Here, struct is the keyword, employee is the tag name of structure; id, name and salary are the members or fields of the structure. Let's understand it by the diagram given below:



Declaring structure variable

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

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

1st way:

Let's see the example to declare a 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;

2nd way:

Let's see another way to declare variables at the time of defining structure.

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

Which approach is good?

If no. of variables are not fixed, use the 1st approach. It provides you flexibility to declare the structure variable many times. If no. of variables are fixed, use the 2nd approach. It saves your code to declare variable in main() fuction.

Accessing members of structure

There are two ways to access structure members:

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

Let's see the code to access the id member of p1 variable by . (member) operator.

p1.id

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
```

NESTED STRUCTURES

Nested structure in the C language can have another structure as a member. There are two ways to define nested structure in c language:

1. By separate structure
2. By Embedded structure

Separate structure

We can create 2 structures, but dependent structure should be used inside the main structure as a member. Let's see the code of the nested structure.

```
struct Date
{
	int dd;
	int mm;
	int yyyy;
};
struct Employee
{
	int id;
	char name[20];
	struct Date doj;
}emp1;
```

As you can see, doj (date of joining) is the variable of type Date. Here doj is used as a member in Employee structure. In this way, we can use Date structure in many structures.

Embedded structure

We can define structure within the structure also. It requires less code than the previous way. But it can't be used in many structures.

```
struct Employee
{
	int id;
	char name[20];
	struct Date
	{
		int dd;
		int mm;
		int yyyy;
	}doj;
}emp1;
```

Accessing Nested Structure

We can access the member of nested structure by Outer_Structure. Nested_Structure.member as given below:

```
	e1.doj.dd
	e1.doj.mm
```