C Nested Structure

- Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.
- The structure variables can be a normal structure variable or a pointer variable to access the data.

The structure can be nested in the following ways.

1. By separate structure
2. By Embedded structure

1) Separate structure

Here, we create two structures, but the dependent structure should be used inside the main structure as a member. Consider the following example.

```
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.

2) Embedded structure

The embedded structure enables us to declare the structure inside the structure. Hence, it requires less line of codes but it can not be used in multiple data structures. Consider the following example.

```
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 the nested structure by Outer_Structure.Nested_Structure.member as given below:

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

C Nested Structure example

Let's see a simple example of the nested structure in C language.

```
#include <stdio.h>
```

```
#include <string.h>
struct Employee
{
  int id;
  char name[20];
  struct Date
   {
     int dd;
     int mm;
     int yyyy;
   }doj;
}e1;
int main( )
{
  //storing employee information
  e1.id=101;
  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
  e1.doj.dd=10;
  e1.doj.mm=11;
  e1.doj.yyyy=2014;

  //printing first employee information
  printf( "employee id : %d\n", e1.id);
```

```
    printf( "employee name : %s\n", e1.name);

    printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj
.mm,e1.doj.yyyy);

    return 0;

}
```

Self-referential structure

The self-referential structure is a structure that points to the same type of structure. It contains one or more pointers that ultimately point to the same structure.

- Structures are a **user-defined** data structure type in C and C++.
- The main benefit of creating structure is that it can hold the different predefined data types.
- It is initialized with the struct keyword and the structure's name followed by this struct keyword.
- We can easily take the different data types like **integer value, float, char,** and others within the same structure.
- It reduces the complexity of the program.
- Using a single structure can hold the values and data set in a single place.
- In the C++ programming language, placing struct keywords is optional or not mandatory, but it is mandatory in the C programming language.
- Self-referential structure plays a very important role in creating other data structures like Linked list.
- In this type of structure, the object of the same structure points to the same data structure and refers to the data types of the same structure.
- It can have one or more pointers pointing to the same type of structure as their member.
- The self-referential structure is widely used in dynamic data structures such as **trees, linked lists,** etc.

Array of Structure

An array having structure as its base type is known as an **array of structure**. To create an array of structure, first structure is declared and then array of structure is declared just like an ordinary array.

Ex.

If struture is declared like:

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

Then an array of structure can be created like:

```
struct employee emp[10]; /* This is array of structure
```

Ex.

Let's see an example of an array of structures that stores information of 5 students and prints it.

```
#include<stdio.h>

#include <string.h>

struct student{

int rollno;

char name[10];

};

int main(){

int i;
```

```
struct student st[5];
printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
scanf("%d",&st[i].rollno);
printf("\nEnter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
  return 0;
}
```

• typedef in C

The **typedef** is a keyword used in C programming to provide some meaningful names to the already existing variable in the C program. It behaves similarly as we define the alias for the commands. In short, we can say that this keyword is used to redefine the name of an already existing variable.

Syntax of typedef

1. typedef <existing_name> <alias_name>

In the above syntax, '**existing_name**' is the name of an already existing variable while '**alias name**' is another name given to the existing variable.

Example

Let's understand through a simple example.

```
#include <stdio.h>

int main()

{

typedef unsigned int unit;

unit i,j;

i=10;

j=20;

printf("Value of i is :%d",i);

printf("\nValue of j is :%d",j);

return 0;

}
```

Output

```
Value of i is :10
Value of j is :20
```