Structure within a structure, structure pointer

- 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. You can learn below concepts in this section.

1. Structure within structure in C using normal variable
2. Structure within structure in C using pointer variable

1. Structure within structure in C using normal variable:

- This program explains how to use structure within structure in C using normal variable. “student_college_detail’ structure is declared inside “student_detail” structure in this program. Both structure variables are normal structure variables.
- Please note that members of “student_college_detail” structure are accessed by 2 dot(.) operator and members of “student_detail” structure are accessed by single dot(.) operator.

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

struct student_college_detail
{
    int college_id;
    char college_name[50];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail clg_data;
}stu_data;

int main()
{
    struct student_detail stu_data = {1, "Raju", 90.5, 71145,
                                      "Anna University"};
    printf(" Id is: %d \n", stu_data.id);
    printf(" Name is: %s \n", stu_data.name);
    printf(" Percentage is: %f \n\n", stu_data.percentage);

    printf(" College Id is: %d \n",
                    stu_data.clg_data.college_id);
    printf(" College Name is: %s \n",
                    stu_data.clg_data.college_name);
```

```
   return 0;
}
```

Output:

```
Id is: 1
Name is: Raju
Percentage is: 90.500000

College Id is: 71145
College Name is: Anna University
```

Structure within structure in C using pointer variable:

- This program explains how to use structure within structure in C using pointer variable. “student_college_detail’ structure is declared inside “student_detail” structure in this program. one normal structure variable and one pointer structure variable is used in this program.
- Please note that combination of .(dot) and ->(arrow) operators are used to access the structure member which is declared inside the structure.

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

struct student_college_detail
{
   int college_id;
   char college_name[50];
};

struct student_detail
{
   int id;
   char name[20];
   float percentage;
   // structure within structure
   struct student_college_detail clg_data;
}stu_data, *stu_data_ptr;

int main()
{
  struct student_detail stu_data = {1, "Raju", 90.5, 71145,
                                    "Anna University"};
```

```
    stu_data_ptr = &stu_data;

    printf(" Id is: %d \n", stu_data_ptr->id);
    printf(" Name is: %s \n", stu_data_ptr->name);
    printf(" Percentage is: %f \n\n",
                    stu_data_ptr->percentage);

    printf(" College Id is: %d \n",
                    stu_data_ptr->clg_data.college_id);
    printf(" College Name is: %s \n",
                  stu_data_ptr->clg_data.college_name);

    return 0;
}
```

Output:

```
Id is: 1
Name is: Raju
Percentage is: 90.500000

College Id is: 71145
College Name is: Anna University
```

Pointer to a Structure in C

We have already learned that a pointer is a variable which points to the address of another variable of any data type like int, char, float etc. Similarly, we can have a pointer to structures, where a pointer variable can point to the address of a structure variable. Here is how we can declare a pointer to a structure variable.

```
struct dog

{

   char name[10];

   char breed[10];

   int age;
```

```
    char color[10];
};

struct dog spike;

// declaring a pointer to a structure of type struct dog
struct dog *ptr_dog
```

This declares a pointer ptr_dog that can store the address of the variable of type struct dog. We can now assign the address of variable spike to ptr_dog using & operator.

```
ptr_dog = &spike;
```

Now ptr_dog points to the structure variable spike.

Accessing members using Pointer #

There are two ways of accessing members of structure using pointer:

Using indirection (*) operator and dot(.) operator.

1 .Using arrow (->) operator or membership operator.

Let's start with the first one.

Using Indirection (*) Operator and Dot(.) Operator #

At this point ptr_dog points to the structure variable spike, so by dereferencing it we will get the contents of the spike. This means spike and *ptr_dog are functionally equivalent. To access a member of structure write *ptr_dog followed by a dot(.) operator, followed by the name of the member. For example:

(*ptr_dog).name - refers to the name of dog (*ptr_dog).breed - refers to the breed of dog

and so on.

Parentheses around *ptr_dog are necessary because the precedence of dot(.) operator is greater than that of indirection (*) operator.

Using arrow operator (->) #

The above method of accessing members of the structure using pointers is slightly confusing and less readable, that's why C provides another way to access members using the arrow (->) operator. To access members using arrow (->) operator write pointer variable followed by -> operator, followed by name of the member.

ptr_dog->name - refers to the name of dog

ptr_dog->breed - refers to the breed of dog

and so on.

Here we don't need parentheses, asterisk(*) and dot(.) operator. This method is much more readable and intuitive.

We can also modify the value of members using pointer notation.

strcpy(ptr_dog->name, "new_name");

Here we know that the name of the array (ptr_dog->name) is a constant pointer and points to the 0th element of the array. So we can't assign a new string to it using assignment operator(=), that's why strcpy() function is used.

--ptr_dog->age;

In the above expression precedence of arrow operator (->) is greater than that of prefix decrement operator (--), so first -> operator is applied in the expression then its value is decremented by 1.

The following program demonstrates how we can use a pointer to structure.

```
#include<stdio.h>

struct dog
{
    char name[10];
    char breed[10];
    int age;
    char color[10];
};

int main()
{
    struct dog my_dog = {"tyke", "Bulldog", 5, "white"};
    struct dog *ptr_dog;
    ptr_dog = &my_dog;

    printf("Dog's name: %s\n", ptr_dog->name);
    printf("Dog's breed: %s\n", ptr_dog->breed);
    printf("Dog's age: %d\n", ptr_dog->age);
    printf("Dog's color: %s\n", ptr_dog->color);
    // changing the name of dog from tyke to jack
    strcpy(ptr_dog->name, "jack");

    // increasing age of dog by 1 year
    ptr_dog->age++;

    printf("Dog's new name is: %s\n", ptr_dog->name);
    printf("Dog's age is: %d\n", ptr_dog->age);

    // signal to operating system program ran fine
    return 0;
}
```

Expected Output:

```
Dog's name: tyke
Dog's breed: Bulldog
Dog's age: 5
Dog's color: white

After changes

Dog's new name is: jack
Dog's age is: 6
```