STRUCTURE in C Programming…

Structure

- A structure is a collection of variables of different data types under a single name.
- The variables are called members of the structure.
- The structure is also called a user-defined data type.

Defining a Structure

- Syntax:

```
struct structure_name
{
 data_type1 member_variable1;
  data_type2 member_variable2;
   …………………………………;
data_typeN member_variableN;
};
```

Note: The members of a structure do not occupy memory until they are associated with a structure_variable.

Example

```
struct student
        {
        char name[20];
        int roll_no;
        float marks;
        char gender;
        long int phone_no;
        };
```

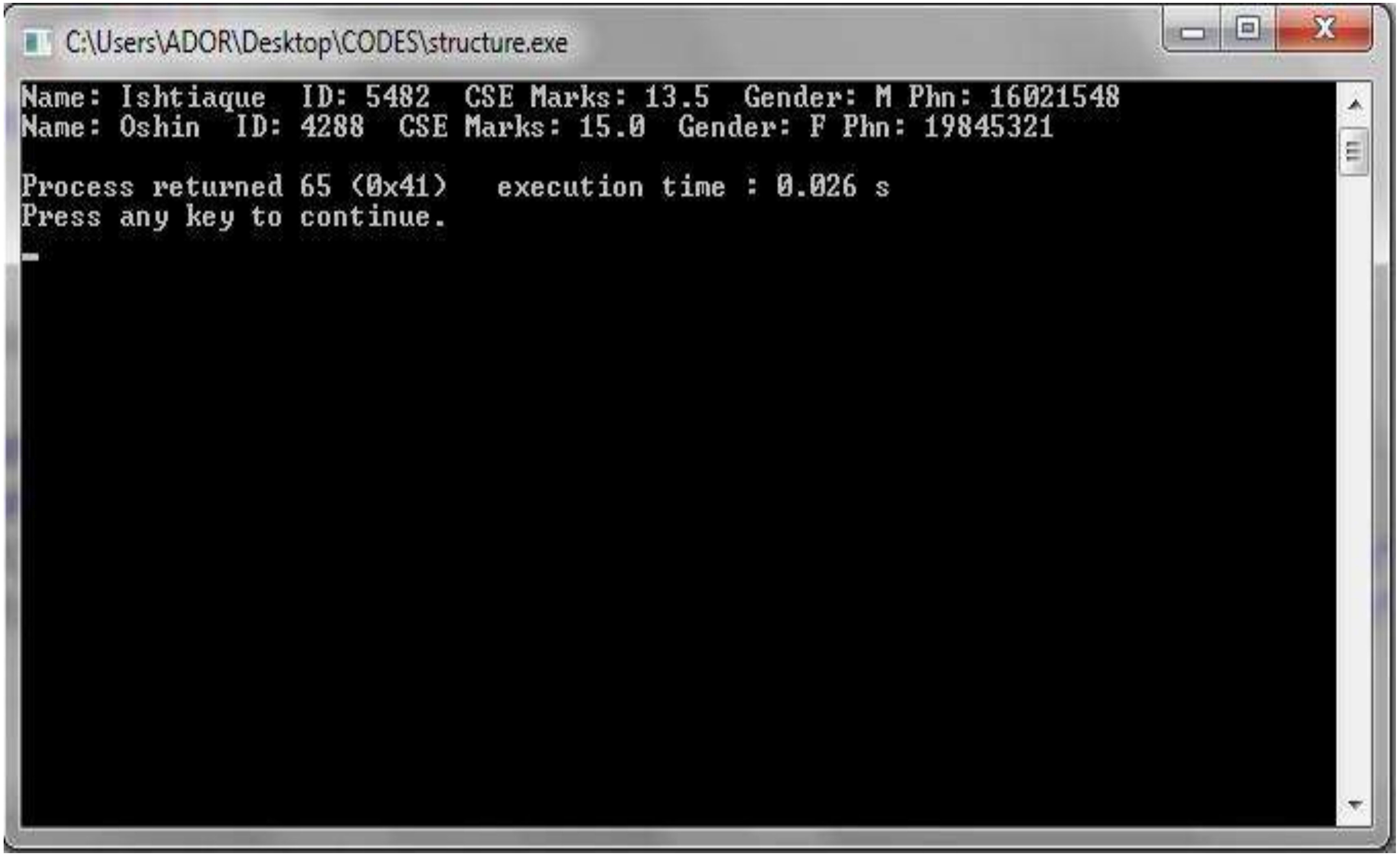
•Multiple variables of *struct student* type can be declared as:

```
struct student st1, st2, st3;
```

Defining a structure…

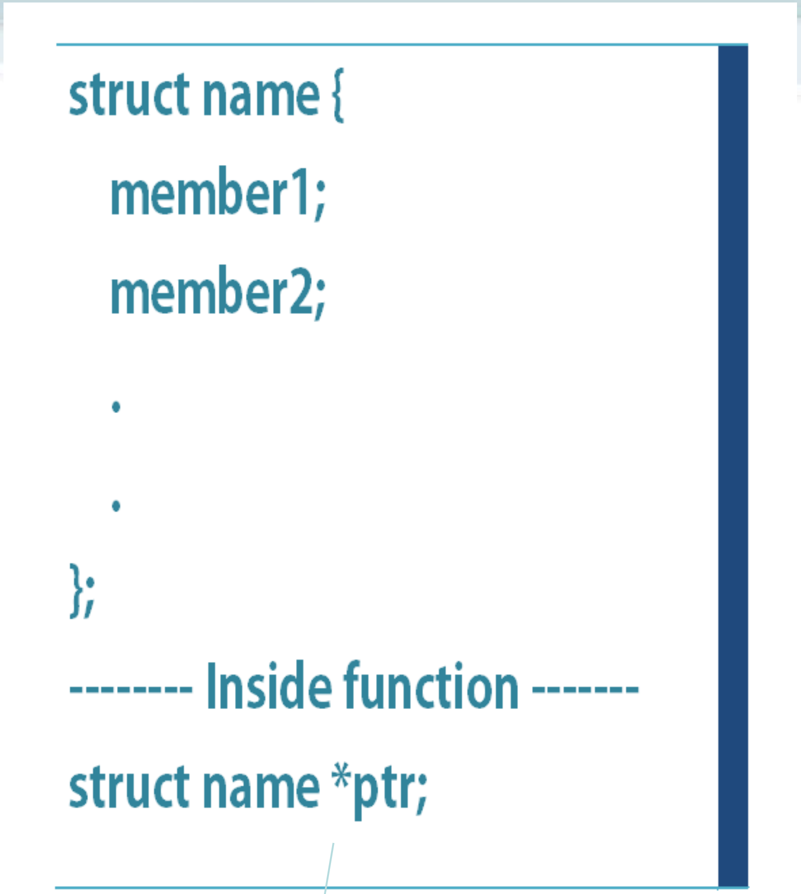
- Each variable of structure has its own copy of member variables.
- The member variables are accessed using the dot (.) operator or member operator.
- For example: *st1.name* is member variable *name* of *st1* structure variable while *st3.gender* is member variable *gender* of *st3* structure variable.

```
struct student
     {
     char name[20];
     int ID;
     float CSE_marks;
     char gender;
     long int phone_no;
     };

void main()
{
   struct  student  st1={"Ishtiaque",5482,13.5,'M',16021548};
   struct  student  st2={"Oshin",4288,15,'F',19845321};

   printf ("Name: %s  ID: %d  CSE Marks: %.1f  Gender: %c Phn: %d
    \n",st1.name,st1.ID,st1.CSE_marks,st1.gender,st1.phone_no);

   printf ("Name: %s  ID: %d  CSE Marks: %.1f  Gender: %c Phn: %d
    \n",st2.name,st2.ID,st2.CSE_marks,st2.gender,st2.phone_no);
}
```



```
Name: Ishtiaque  ID: 5482  CSE Marks: 13.5  Gender: M Phn: 16021548
Name: Oshin  ID: 4288  CSE Marks: 15.0  Gender: F Phn: 19845321

Process returned 65 (0x41)   execution time : 0.026 s
Press any key to continue.
```

Declarationofpointers



```
struct name {
   member1;
   member2;
   .
   .
};
-------- Inside function -------
struct name *ptr;
```

Structure'smemberthrough pointercanbeusedintwoways:

- Referencing pointer to another address to access memory
- Using dynamic memory allocation

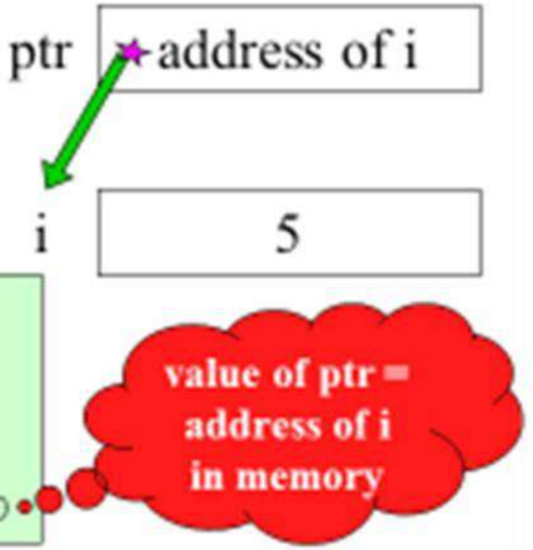
What actually *ptr* is?

- **ptr** is a variable storing **an address**
- ptr is **NOT** storing the actual value of i

```
int i = 5;
int *ptr;
ptr = &i;
printf("i = %d\n", i);
printf("*ptr = %d\n", *ptr);
printf("ptr = %p\n", ptr);
```

Output:

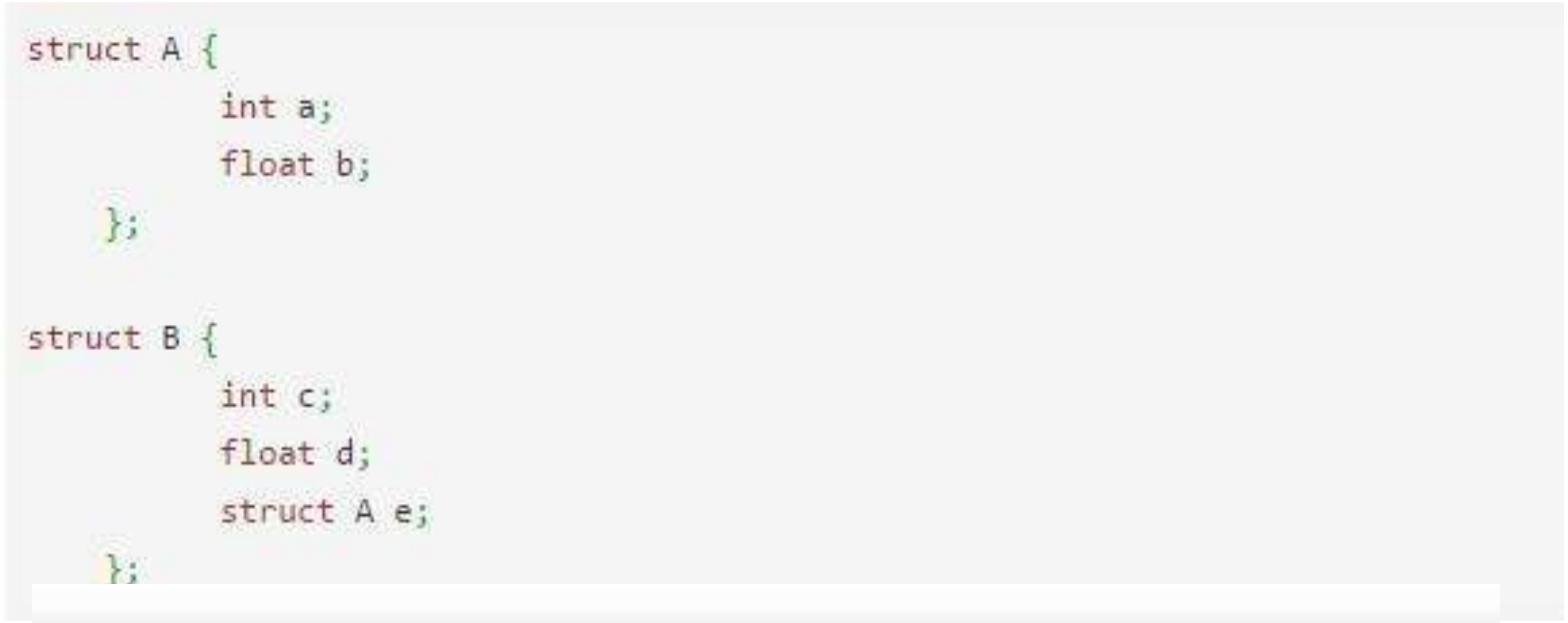
```
i = 5
*ptr = 5
ptr = effff5e0
```



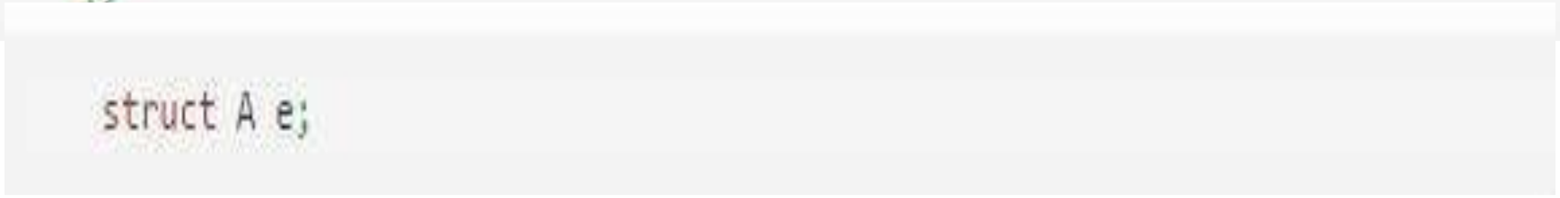
What is Nested Structure

- **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.**
- **Nested Structures are allowed in C Programming Language.**
- **We can write one Structure inside another structure as** member **of another structure.**

Let’s see the structure declaration below.



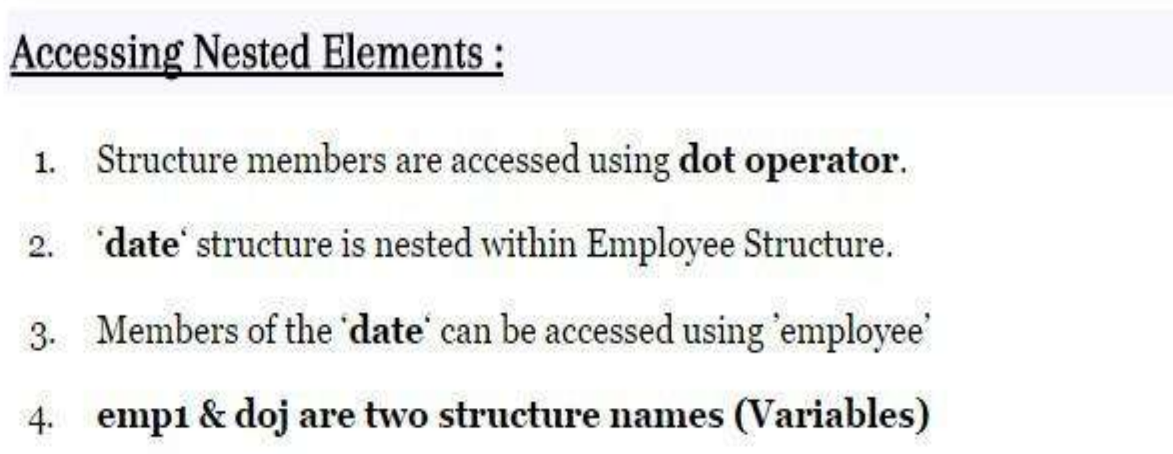
```
struct A {
          int a;
          float b;
    };

struct B {
          int c;
          float d;
          struct A e;
    };
```



```
struct A e;
```

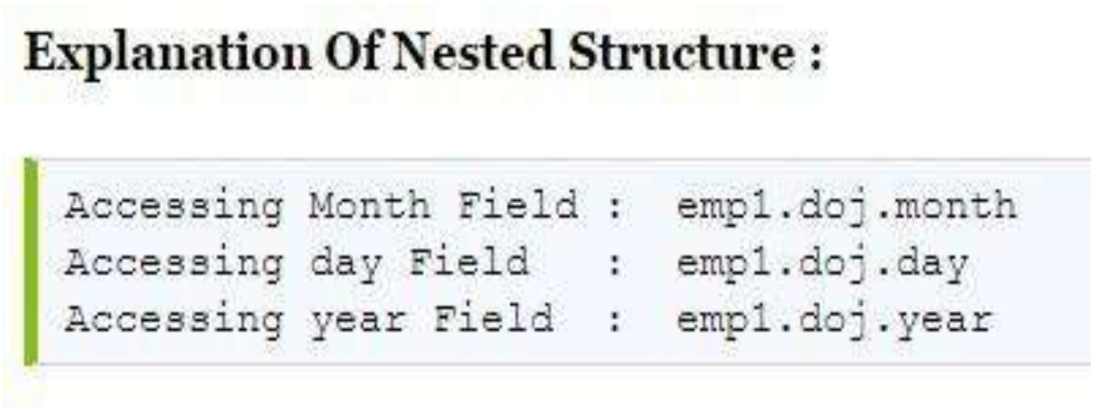
Way 1: Declare two separate structures

```
 struct date
  {
 int date;
        int month;
 int year;
   };
    struct Employee
{
 char ename[20];
            int ssn;
 float salary;

     struct date doj;
}
emp1;
```



Accessing Nested Elements :

1. Structure members are accessed using **dot operator**.
2. '**date**' structure is nested within Employee Structure.
3. Members of the '**date**' can be accessed using 'employee'
4. **emp1 & doj are two structure names (Variables)**

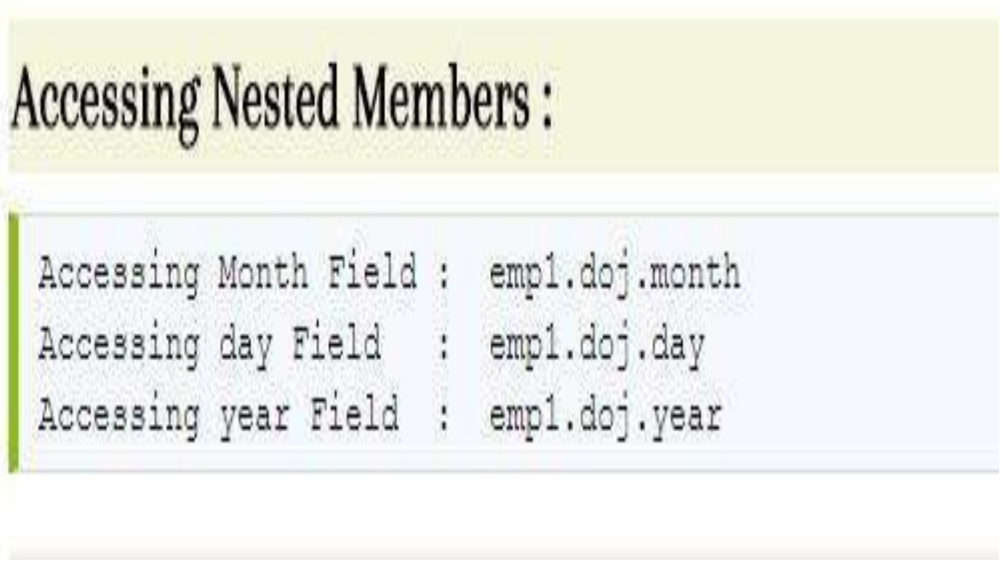


Explanation Of Nested Structure :

```
Accessing Month Field :  emp1.doj.month
Accessing day Field   :  emp1.doj.day
Accessing year Field  :  emp1.doj.year
```

Way 2 : Declare Embedded structures

```
struct Employee
 {
 char ename[20];
 int ssn;
 float salary;
 struct date
{
 int date;
int month;
 int year;
 }
doj;
 }
emp1;
```


Accessing Nested Members :

```
Accessing Month Field  :  emp1.doj.month
Accessing day Field    :  emp1.doj.day
Accessing year Field   :  emp1.doj.year
```

Function and Structure

We will consider four cases here:

- *Passing the individual member to functions*
- *Passing whole structure to functions*
- *Passing structure pointer to functions*
- *Passing array of structure to functions*

Passing the individual member to functions

- ***Structure members can be passed to functions as actual arguments in function call like ordinary variables.***

PASSING WHOLE STRUCTURE TO FUNCTIONS

- ***Whole structure can be passed to a function by the syntax:***

  function _ name (structure _ variable _ name);

- ***The called function has the form:***

  return _ type function _ name (struct tag _ name structure _ variable _ name)

  {

  …………;

  }