

## UNIT-4/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.

Let's look at the first approach in detail.

```
1. #include<stdio.h>
2. void main ()
3. {
4.     char names[2][10],dummy; // 2-
    dimensionaal character array names is used to store the names of the students
5.     int roll_numbers[2],i;
6.     float marks[2];
7.     for (i=0;i<3;i++)
8.     {
9.
10.    printf("Enter the name, roll number, and marks of the student %d",i+1);
11.    scanf("%s %d %f",&names[i],&roll_numbers[i],&marks[i]);
12.    scanf("%c",&dummy); // enter will be stored into dummy character at each iteration
13. }
14. printf("Printing the Student details ...\n");
15. for (i=0;i<3;i++)
16. {
17.    printf("%s %d %f\n",names[i],roll_numbers[i],marks[i]);
18. }
```

19.}

### Output:

```
Enter the name, roll number, and marks of the student 1Arun 90 91
Enter the name, roll number, and marks of the student 2Varun 91 56
Enter the name, roll number, and marks of the student 3Sham 89 69

Printing the Student details...
Arun 90 91.000000
Varun 91 56.000000
Sham 89 69.000000
```

The above program may fulfill our requirement of storing the information of an entity student. However, the program is very complex, and the complexity increase with the amount of the input. The elements of each of the array are stored contiguously, but all the arrays may not be stored contiguously in the memory. C provides you with an additional and simpler approach where you can use a special data structure, i.e., structure, in which, you can group all the information of different data type regarding an entity.

### 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 can simulate the use of classes and templates as it can store various information

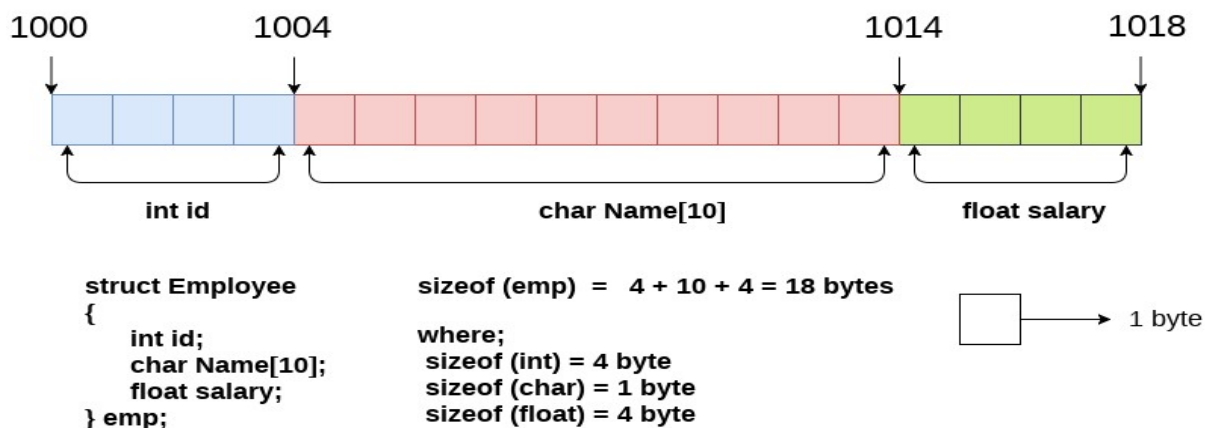
The **struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

1. **struct** structure\_name
2. {
3.     data\_type member1;
4.     data\_type member2;
5.     data\_type memberN;
6. };

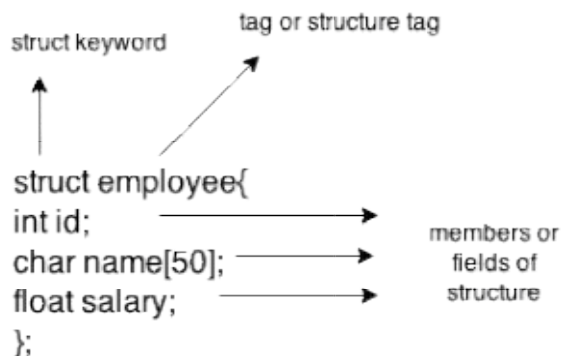
Let's see the example to define a structure for an entity employee in c.

1. **struct** employee
2. { **int** id;
3. **char** name[20];
4. **float** salary;
5. };

The following image shows the memory allocation of the structure employee that is defined in the above example.



Here, **struct** is the keyword; **employee** is the name of the 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 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:**

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

1. **struct** employee
2. { **int** id;
3. **char** name[50];
4. **float** salary;
5. };

Now write given code inside the main() function.

1. **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:**

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

1. **struct** employee
2. { **int** id;
3. **char** name[50];
4. **float** salary;
5. }e1,e2;

## Accessing members of the 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`

### C Structure example

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

```
1. #include<stdio.h>
2. #include <string.h>
3. struct employee
4. {   int id;
5.     char name[50];
6. }e1; //declaring e1 variable for structure
7. int main( )
8. {
9.     //store first employee information
10.    e1.id=101;
11.    strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
12.    //printing first employee information
13.    printf( "employee 1 id : %d\n", e1.id);
14.    printf( "employee 1 name : %s\n", e1.name);
15. return 0;
16. }
```

Output:

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

Let's see another example of the structure in [C language](#) to store many employees information.

```
1. #include<stdio.h>
2. #include <string.h>
3. struct employee
4. { int id;
5.   char name[50];
6.   float salary;
7. }e1,e2; //declaring e1 and e2 variables for structure
8. int main( )
9. {
10. //store first employee information
11. e1.id=101;
12. strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
13. e1.salary=56000;
14.
15. //store second employee information
16. e2.id=102;
17. strcpy(e2.name, "James Bond");
18. e2.salary=126000;
19.
20. //printing first employee information
21. printf( "employee 1 id : %d\n", e1.id);
22. printf( "employee 1 name : %s\n", e1.name);
23. printf( "employee 1 salary : %f\n", e1.salary);
24.
25. //printing second employee information
26. printf( "employee 2 id : %d\n", e2.id);
27. printf( "employee 2 name : %s\n", e2.name);
28. printf( "employee 2 salary : %f\n", e2.salary);
29. return 0;
30.}
```

Output:

```
employee 1 id : 101
employee 1 name : Sonoo Jaiswal
employee 1 salary : 56000.000000
employee 2 id : 102
employee 2 name : James Bond
employee 2 salary : 126000.000000
```

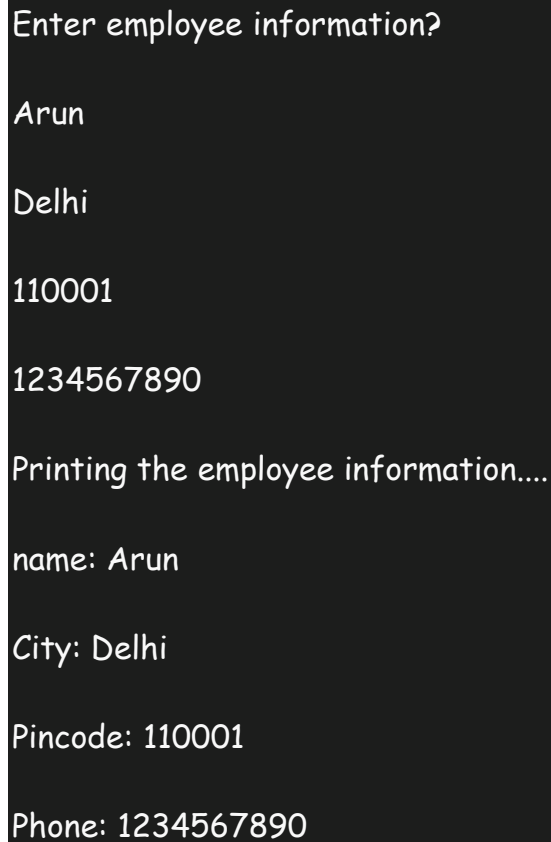
## Nested Structure in C

C provides us the feature of nesting one structure within another structure by using which, complex data types are created. For example, we may need to store the address of an entity employee in a structure. The attribute address may also have the subparts as street number, city, state, and pin code. Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee. Consider the following program.

```
1. #include<stdio.h>
2. struct address
3. {
4.     char city[20];
5.     int pin;
6.     char phone[14];
7. };
8. struct employee
9. {
10.    char name[20];
11.    struct address add;
12.};
13. void main ()
14. {
15.    struct employee emp;
16.    printf("Enter employee information?\n");
17.    scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
```

```
18. printf("Printing the employee information....\n");
19. printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,e
    mp.add.pin,emp.add.phone);
20.}
```

### Output

A screenshot of a terminal window with a dark background. The text is white and shows the output of a C program. It starts with a prompt 'Enter employee information?' followed by four lines of input: 'Arun', 'Delhi', '110001', and '1234567890'. Then, it prints 'Printing the employee information....' followed by four lines of formatted output: 'name: Arun', 'City: Delhi', 'Pincode: 110001', and 'Phone: 1234567890'.

```
Enter employee information?
Arun
Delhi
110001
1234567890
Printing the employee information....
name: Arun
City: Delhi
Pincode: 110001
Phone: 1234567890
```

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.

1. **struct** Date



```
2. {
3.     int dd;
4.     int mm;
5.     int yyyy;
6. };
7. struct Employee
8. {
9.     int id;
10.    char name[20];
11.    struct Date doj;
12.}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.

```
1. struct Employee
2. {
3.     int id;
4.     char name[20];
5.     struct Date
6.     {
7.         int dd;
8.         int mm;
9.         int yyyy;
10.    }doj;
11.}emp1;
```

## Accessing Nested Structure

We can access the member of the nested structure by Outer\_Structure.Nested\_Structure.member as given below:

1. e1.doj.dd
2. e1.doj.mm
3. e1.doj.yyyy

## C Nested Structure example

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

1. #include <stdio.h>
2. #include <string.h>
3. **struct** Employee
4. {
5.   **int** id;
6.   **char** name[20];
7.   **struct** Date
8.   {
9.     **int** dd;
10.    **int** mm;
11.    **int** yyyy;
12.   }doj;
13. }e1;
14. **int** main( )
15. {
16.   //storing employee information
17.   e1.id=101;
18.   strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
19.   e1.doj.dd=10;
20.   e1.doj.mm=11;
21.   e1.doj.yyyy=2014;

```
22. //printing first employee information
23. printf( "employee id : %d\n", e1.id);
24. printf( "employee name : %s\n", e1.name);
25. printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj.
    mm,e1.doj.yyyy);
26. return 0;
27.}
```

Output:

```
employee id : 101
employee name : Sonoo Jaiswal
employee date of joining (dd/mm/yyyy) : 10/11/2014
```

## What is a Union?

In "c," programming union is a user-defined data type that is used to store the different data type's values. However, in the union, one member will occupy the memory at once. In other words, we can say that the size of the union is equal to the size of its largest data member size. Union offers an effective way to use the same memory location several times by each data member. The **union** keyword is used to define and create a union data type.

### Syntax of Union

```
1. union [union name]
2. {
3.     type member_1;
4.     type member_2;
5.     ...
6.     type member_n;
7. };
```

### Example

```
1. union employee
```

```
2. {  
3.   string name;  
4.   string department;  
5.   int phone;  
6.   string email;  
7. };
```

### Below is the declaration for a Union in C:

Union declaration

```
union tagname  
{  
    int a;  
    char b;  
};
```

Here, **union** is the keyword to declare a union, **tagname** is the union name, **a** and **b** are the members of the union **tagname**.

Union variable/object declaration

Now we should create an object for the union in order to access the elements inside it. Below is how we can do that:

```
union tagname object;
```

Here, **object** is the union variable name, that will be used to access the union elements.

### Difference between Structure and Union

Let's summarize the above discussed topic about the Struct and Union in the form of a table that highlight the differences between structure and union:

Struct	Union
The struct keyword is used to define a structure.	The union keyword is used to define union.
When the variables are declared in a structure, the compiler allocates memory to each variables member. The size of a structure is equal or greater to the sum of the sizes of each data member.	When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size.
Each variable member occupied a unique memory space.	Variables members share the memory space of the largest size variable.
Changing the value of a member will not affect other variables members.	Changing the value of one member will also affect other variables members.
Each variable member will be assessed at a time.	Only one variable member will be assessed at a time.
We can initialize multiple variables of a structure at a time.	In union, only the first data member can be initialized.
All variable members store some value at any point in the program.	Exactly only one data member stores a value at any particular instance in the program.
The structure allows initializing multiple variable members at once.	Union allows initializing only one variable member at once.
It is used to store different data type values.	It is used for storing one at a time from different data type values.
It allows accessing and retrieving any data member at a time.	It allows accessing and retrieving any one data member at a time.