Structure in C

Topics:

1) **Introduction and Declaration of Structures**
2) **Nested Structures**
3) **Operations on Structures**
4) **Array of Structures**

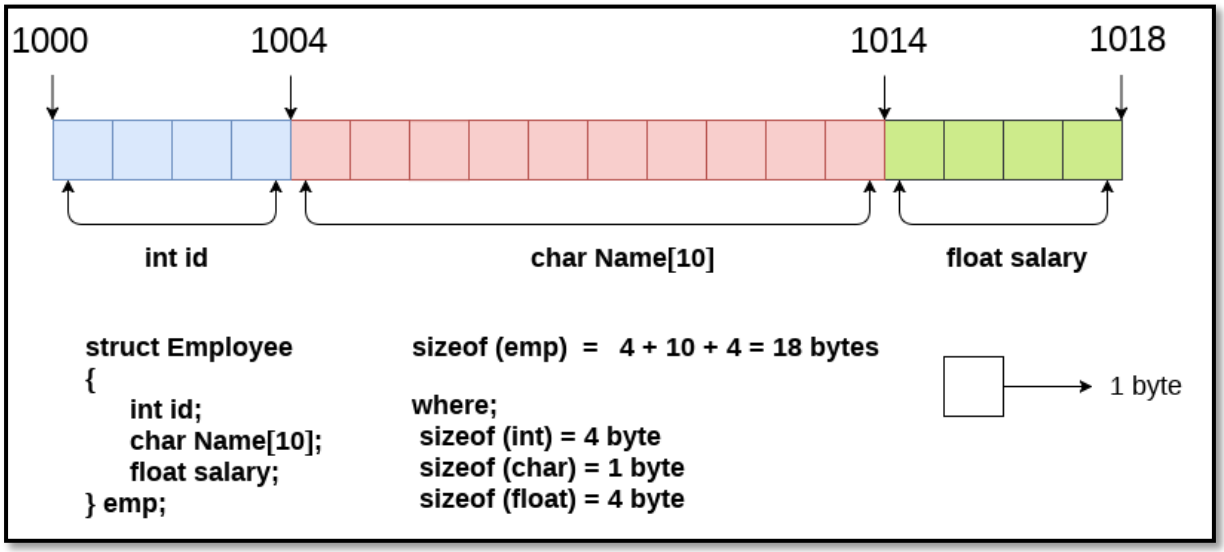
- Arrays allow to define type of variables that can hold several data items of the same kind. Similarly, **structure** is another user defined data type available in C that allows to combine data items of different kinds.
- Structures are used to represent a record.
- Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures 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.

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

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

```
struct employee
{   int id;
    char name[10];
    float salary;
};
```

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



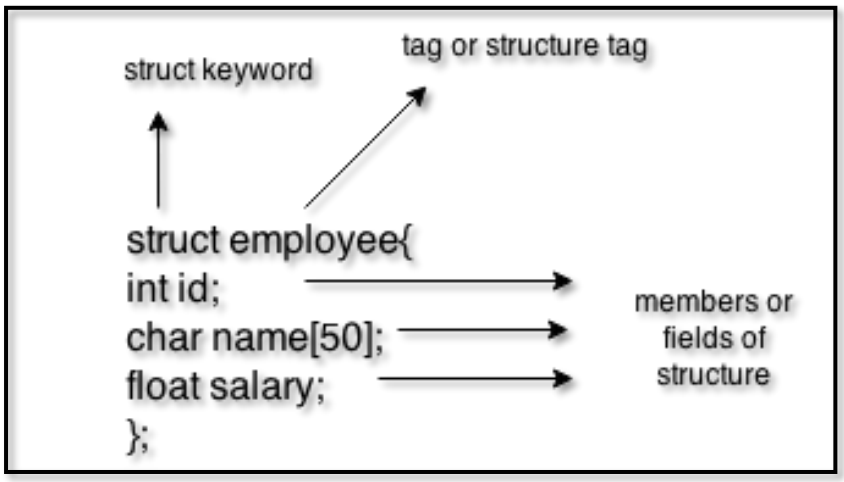
Here,

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.

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

Now write given code inside the main() function.

```
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.

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

Which approach is good

If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.

If no. of variables are fixed, use 2nd approach. It saves your code to declare a variable in main() function.

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.

```
1. p1.id
```

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.
8. int main( )
9. {
10.   //store first employee information
11.   e1.id=101;
12.   strcpy(e1.name, "Sridhar Iyer");//copying string into char array
13.
14.   //printing first employee information
15.   printf( "employee 1 id : %d\n", e1.id);
16.   printf( "employee 1 name : %s\n", e1.name);
17. return 0;
18. }
```

Output:

```
employee 1 id : 101
employee 1 name : Sridhar Iyer
```

Let's see another example of the structure in C language to store many employees information.

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;  //declaring e1 and e2 variables for structure

int main( )
{
  //store first employee information
  e1.id=101;
  strcpy(e1.name, "Harry Potter");//copying string into char array
  e1.salary=56000;

 //store second employee information
  e2.id=102;
  strcpy(e2.name, "James Bond");
  e2.salary=126000;

  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
  printf( "employee 1 salary : %f\n", e1.salary);

  //printing second employee information
  printf( "employee 2 id : %d\n", e2.id);
  printf( "employee 2 name : %s\n", e2.name);
  printf( "employee 2 salary : %f\n", e2.salary);
  return 0;
}
```

Output:

employee 1 id : 101

employee 1 name : Harry Potter

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.
9. struct employee
10. {
11.    char name[20];
12.    struct address add;
13. };
14.
15. void main ()
16. {
17.    struct employee emp;
18.    printf("Enter employee information?\n");
19.    scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.pho
    ne);
20.    printf("Printing the employee information....\n");
21.    printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.
    city,emp.add.pin,emp.add.phone);
22. }
```

Output

```
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.
15. int main( )
16. {
17.   //storing employee information
18.   e1.id=101;
19.   strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
20.   e1.doj.dd=10;
21.   e1.doj.mm=11;
22.   e1.doj.yyyy=2014;
23.
24.   //printing first employee information
25.   printf( "employee id : %d\n", e1.id);
26.   printf( "employee name : %s\n", e1.name);
27.   printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj.m
      m,e1.doj.yyyy);
28.   return 0;
29. }
```

Output:

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

Passing structure to function

Just like other variables, a structure can also be passed to a function. We may pass the structure members into the function or pass the structure variable at once. Consider the following example to pass the structure variable employee to a function display() which is used to display the details of an employee.

```
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 display(struct employee);
14. void main ()
15. {
16.     struct employee emp;
17.     printf("Enter employee information?\n");
18.     scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
19.     display(emp);
20. }
21. void display(struct employee emp)
22. {
23.   printf("Printing the details....\n");
24.   printf("%s %s %d %s",emp.name,emp.add.city,emp.add.pin,emp.add.phone);
25. }
```

Array of structures?

Consider a case, where we need to store the data of 3 students. We can store it by using the structure as given below.

```
1. #include<stdio.h>
2. struct student
3. {
4.     char name[20];
5.     int id;
6.     float marks;
7. };
8. void main()
9. {
10.    struct student s1,s2,s3;
11.    int dummy;
12.    printf("Enter the name, id, and marks of student 1 ");
13.    scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
14.    printf("Enter the name, id, and marks of student 2 ");
15.    scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
16.    printf("Enter the name, id, and marks of student 3 ");
17.    scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
18.    printf("Printing the details....\n");
19.    printf("%s %d %f\n",s1.name,s1.id,s1.marks);
20.    printf("%s %d %f\n",s2.name,s2.id,s2.marks);
21.    printf("%s %d %f\n",s3.name,s3.id,s3.marks);
22. }
```

Output

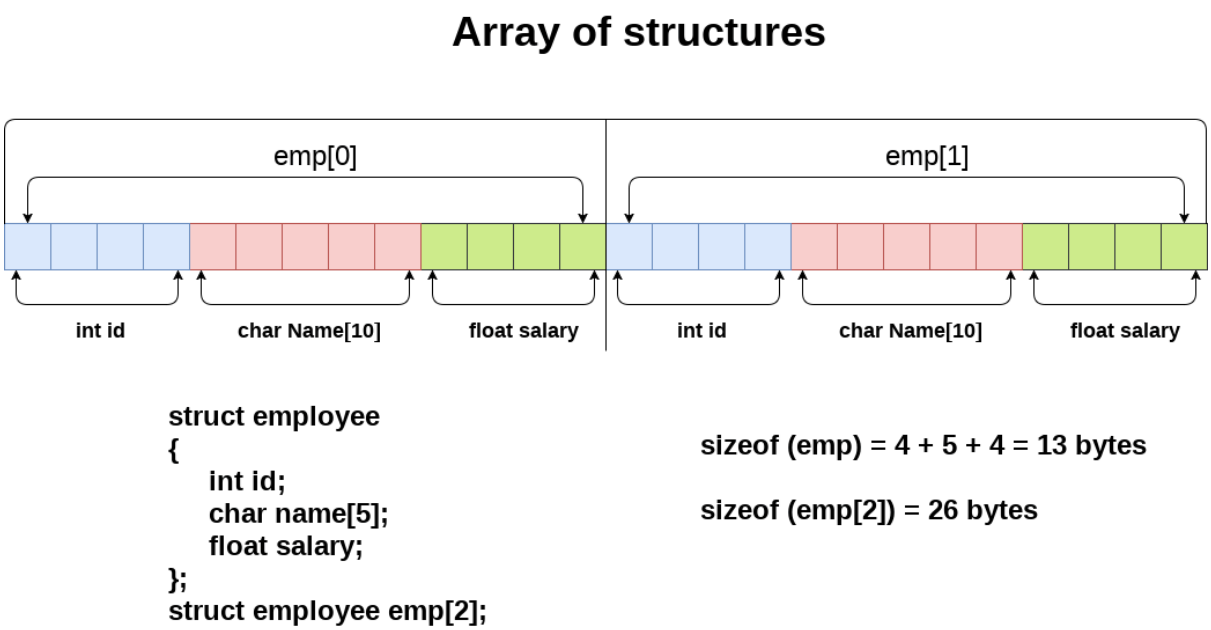
```
Enter the name, id, and marks of student 1 James 90 90
Enter the name, id, and marks of student 2 Adoms 90 90
Enter the name, id, and marks of student 3 Nick 90 90
Printing the details....
James 90 90.000000
Adoms 90 90.000000
Nick 90 90.000000
```

In the above program, we have stored data of 3 students in the structure. However, the complexity of the program will be increased if there are 20 students. In that case, we will have to declare 20 different structure variables and store them one by one. This will always be tough since we will have to declare a variable every time we add a

student. Remembering the name of all the variables is also a very tricky task. However, c enables us to declare an array of structures by using which, we can avoid declaring the different structure variables; instead we can make a collection containing all the structures that store the information of different entities.

Array of Structures in C

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.



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

```
1. #include<stdio.h>
2. #include <string.h>
3. struct student
4. {
5. int rollno;
6. char name[10];
7. };
8. int main()
9. {
10. int i;
```

```
11. struct student st[5];
12. printf("Enter Records of 5 students");
13. for(i=0;i<5;i++){
14. printf("\nEnter Rollno:");
15. scanf("%d",&st[i].rollno);
16. printf("\nEnter Name:");
17. scanf("%s",&st[i].name);
18. }
19. printf("\nStudent Information List:");
20. for(i=0;i<5;i++){
21. printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
22. }
23.    return 0;
24. }
```

Output:

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
Enter Rollno:5
Enter Name:Sarfraz

Student Information List:
Rollno:1, Name:Sonoo
Rollno:2, Name:Ratan
Rollno:3, Name:Vimal
Rollno:4, Name:James
Rollno:5, Name:Sarfraz
```