C 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-dimensioanal 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. {
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 ca; 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. .
6. .
7. data\_type memeberN;
8. };

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.

c structure memory allocation

## 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;

#### **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

## 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 : SonooJaiswal

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;
15. //store second employee information
16. e2.id=102;
17. strcpy(e2.name, "James Bond");
18. e2.salary=126000;
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);
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 : SonooJaiswal

employee 1 salary : 56000.000000

employee 2 id : 102

employee 2 name : James Bond

employee 2 salary : 126000.000000

# C Array of Structures

## Why use an array of structures?

Consider a case, where we need to store the data of 5 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. scanf("%c",&dummy);
15. printf("Enter the name, id, and marks of student 2 ");
16. scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
17. scanf("%c",&dummy);
18. printf("Enter the name, id, and marks of student 3 ");
19. scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
20. scanf("%c",&dummy);
21. printf("Printing the details....\n");
22. printf("%s %d %f\n",s1.name,s1.id,s1.marks);
23. printf("%s %d %f\n",s2.name,s2.id,s2.marks);
24. printf("%s %d %f\n",s3.name,s3.id,s3.marks);
25. }

#### **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 structres 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.

c array 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. **int** rollno;
5. **char** name[10];
6. };
7. **int** main(){
8. **int** i;
9. **struct** student st[5];
10. printf("Enter Records of 5 students");
11. **for**(i=0;i<5;i++){
12. printf("\nEnter Rollno:");
13. scanf("%d",&st[i].rollno);
14. printf("\nEnter Name:");
15. scanf("%s",&st[i].name);
16. }
17. printf("\nStudent Information List:");
18. **for**(i=0;i<5;i++){
19. printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
20. }
21. **return** 0;
22. }

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

# 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,emp.add.pin,emp.add.phone);
20. }

#### **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;  //emp1.doj.dd=12/03/1991
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;
23. //printing first employee information
24. printf( "employee id : %d\n", e1.id);
25. printf( "employee name : %s\n", e1.name);
26. printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj.mm,e1.doj.yyyy);
27. **return** 0;
28. }

Output:

employee id : 101

employee name : SonooJaiswal

employee date of joining (dd/mm/yyyy) : 10/11/2014