STRUCTURE IN C

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.

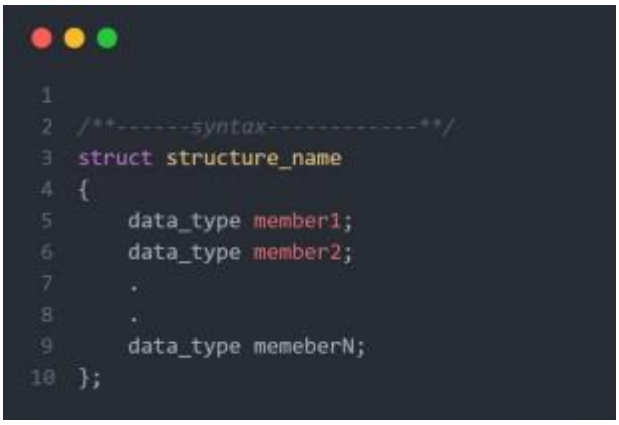
Let's look at the first approach in detail

```c
#include <stdio.h>
void main()
{
    char names[2][10], dummy;
// 2-dimensioonal character array names is used to store the names of the students
    int roll_numbers[2], i;
    float marks[2];
    for (i = 0; i < 3; i++)
    {

        printf("Enter the name, roll number, and marks of the student %d", i + 1);
        scanf("%s %d %f", &names[i], &roll_numbers[i], &marks[i]);
        scanf("%c", &dummy);
// enter will be stored into dummy character at each iteration
    }
    printf("Printing the Student details ...\n");
    for (i = 0; i < 3; i++)
    {
        printf("%s %d %f\n", names[i], roll_numbers[i], marks[i]);
    }
}

```

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.

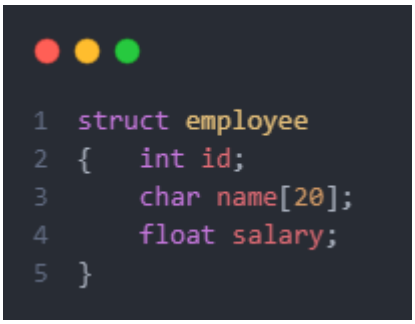
What is Structure Structure

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



```
/**------syntax-----------**/
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[20];
    float salary;
}
```

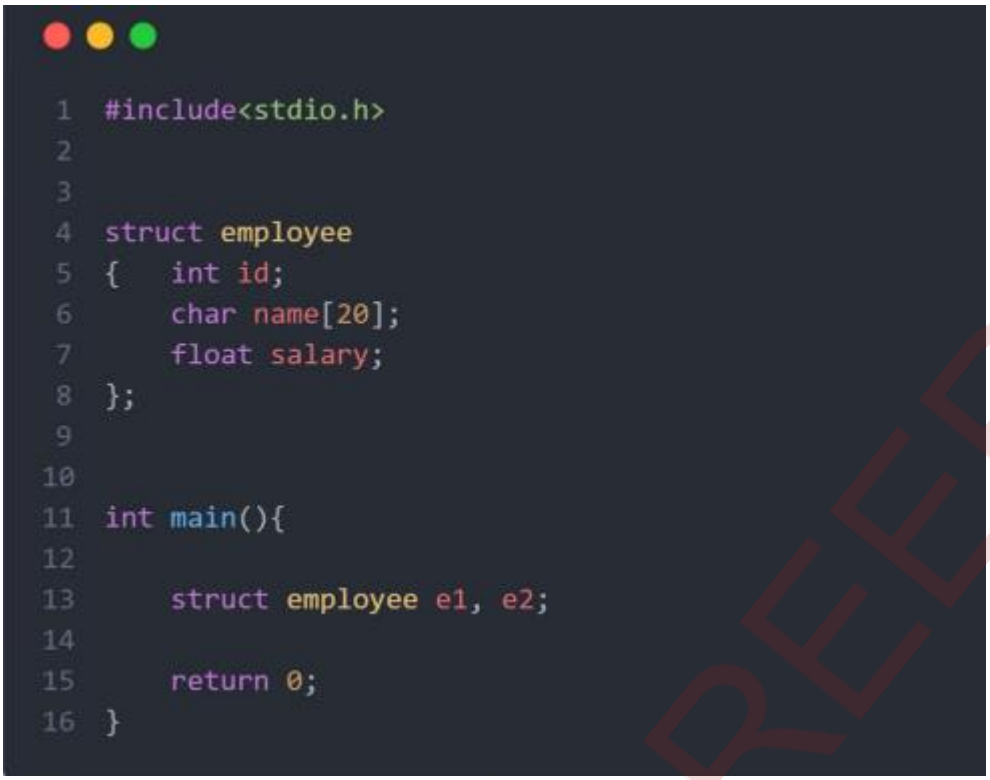
Declaring structure variable

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.



```c
#include<stdio.h>


struct employee
{   int id;
    char name[20];
    float salary;
};


int main(){

    struct employee e1, e2;

    return 0;
}
```

2nd way:

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

```c
#include<stdio.h>


struct employee
{   int id;
    char name[20];
    float salary;
}e1,e2;


int main(){



    return 0;
}

```

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:

By . (member or dot operator)

By -> (structure pointer operator)

C Structure example

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

```c
#include <stdio.h>
#include <string.h>
struct employee
{
    int id;
    char name[50];
    float salary;
} ;
int main()
{
    //declare a structure variable
    struct employee e1= {101,"Rajesh",10000};
    struct employee e2= {102,"Mahesh",15000};

    // printing first employee information
    printf("employee 1 id : %d\n", e1.id);
    printf("employee 1 name : %s\n", e1.name);
    printf("employee 1 salary : %.2f\n", e1.salary);

    return 0;
}
```

C Array of Structure

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.

```c

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

struct student
{
    int roll;
    char name[50];
};

int main()
{
    struct student st1 = {10, "Arnab"};
    printf("\nRoll No:%d,Name:%s", st1.roll, st1.name);

    struct student st2 = {20, "Srijan"};
    printf("\nRoll No:%d,Name:%s", st2.roll, st2.name);

    struct student st3 = {30, "Koushik"};
    printf("\nRoll No:%d,Name:%s", st3.roll, st3.name);

    struct student st4 = {40, "Deepak"};
    printf("\nRoll No:%d,Name:%s", st4.roll, st4.name);

    struct student st5 = {50, "Madhusree"};
    printf("\nRoll No:%d,Name:%s", st5.roll, st5.name);

    return 0;
}
```

OUTPUT

```
Roll No:10,Name:Arnab
Roll No:20,Name:Srijan
Roll No:30,Name:Koushik
Roll No:40,Name:Deepak
Roll No:50,Name:Madhusree
```

In the above program, we have stored data of 5 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
#include <stdio.h>
#include <string.h>

struct student
{
    int roll;
    char name[50];
};

int main()
{
    int i;
    struct student st[5];
    printf("Enter the records of 5 students:\n ");
    for (i = 0; i < 5; i++)
    {
        printf("\n Enter Roll No %d :",i+1);
        scanf("%d", &st[i].roll);
        printf("\n Enter Name%d :",i+1);
        scanf("%s", &st[i].name);
    }

    for (i = 0; i < 5; i++)
    {
        printf("\nRoll No:%d,Name:%s", st[i].roll, st[i].name);
    }

    return 0;
}
```

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.

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.

```c
struct date
{
    int day;
    int month;
    int year;
};

struct employee
{
    int id;
    char name[20];
    struct date doj;
};
```

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.

```
struct Employee
{
    int id;
    char name[20];
    struct Date
     {
       int dd;
       int mm;
       int yyyy;
     };
};
```

C Nested Structure example

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

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

struct Date
{
   int dd;
   int mm;
   int yyyy;
};

struct Employee
{
   int id;
   char name[20];
   struct Date doj;

};

int main( )
{
   //?storing employee information
   struct Employee e1;
   e1.id=101;
   strcpy(e1.name, "Sonoo Jaiswal");//!copying string into char array
   e1.doj.dd=10;
   e1.doj.mm=11;
   e1.doj.yyyy=2014;

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

OUTPUT

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

Pointer to structure

```c
/*
TODO: CREATE A STRUCTURE AS A ARGUMENT OF A FUNCTION
*/

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

//!CRETE A STRUCTURE STUDENT
struct student
{
    char name[100];
    int roll;
    int marks;
};

//!CREATE A FUNCTION TO DISPLAY THE STUDENT DETAILS
/*
    * here we are passing the structure pointer as a argument of the function
    * ptr is the structure pointer
    * n is the number of students
*/
void displayStudent(struct student *ptr, int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf("Student name: %s\n", ptr[i].name);
        printf("Student roll: %d\n", ptr[i].roll);
        printf("Student marks: %d\n", ptr[i].marks);
    }
}

int main()
{
    struct student s[100];                    //?ARRAY OF STRUCTURE
    int n = 0;
    int i;
    //!CREATE A STRUCTURE POINTER
    for (i = 0; i < 2; i++)
    {
        printf("Enter student name: ");
        scanf("%s", s[n].name);
        fflush(stdin);                        //?CLEAR THE BUFFER
        printf("Enter student roll: ");
        scanf("%d", &s[n].roll);
        printf("Enter student marks: ");
        scanf("%d", &s[n].marks);
        n = n + 1;                            //?INCREMENT THE NUMBER OF STUDENTS
    }
    //!DISPLAY THE STUDENT DETAILS
    /*
        * here we are passing the structure variable as a argument of the function
        * s is the structure variable
        * n is the number of students
    */
    displayStudent(s, n);
    return 0;
}

```

Union in C

is a special data type available in C that allows storing different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purposes.

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.