C Structure

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.

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) etc. To store such type of information regarding an entity student, we have the following approaches:

- Construction of individual arrays for storing the attributes such as names, roll numbers, and marks.
- Use a special data structure to store the collection of different data types using structure.

Let's look at the first approach in detail.

```
#include<stdio.h>
   void main ()
   {
     char names[10][10]; // 2-dimensioanal character array names is used to store the names of the students
     int roll_numbers[10],i;
     float marks[10];
     for (int i=0;i<3;i++)
     {
       printf("Enter the name of the student %d:",i+1);
       scanf("%s",&names[i]);
       printf("Enter the roll number of the student %d:",i+1);
       scanf("%d",&roll_numbers[i]);
       printf("Enter the marks of the student %d:",i+1);
       scanf("%f",&marks[i]);
      }
     printf("Printing the Student details ...\n");
     for (int i=0;i<3;i++)
     {
       printf("%s %d %f\n",names[i],roll_numbers[i],marks[i]);
     }
   }
```

OUTPUT

```
Enter the name of the student 1:Hari
Enter the roll number of the student 1:1111056
Enter the marks of the student 1:46
Enter the name of the student 2:Ramu
Enter the roll number of the student 2:1111058
Enter the marks of the student 2:42
```

Enter the name of the student 3:Om
Enter the roll number of the student 3:1111060
Enter the marks of the student 3:35
Printing the Student details ...
Hari 1111056 46.000000
Ramu 1111058 42.000000
Om 1111060 35.000000

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. To define a structure, you must use the struct statement. The **struct** statement defines a new data type, with more than one member. The format of the struct statement is as follows –

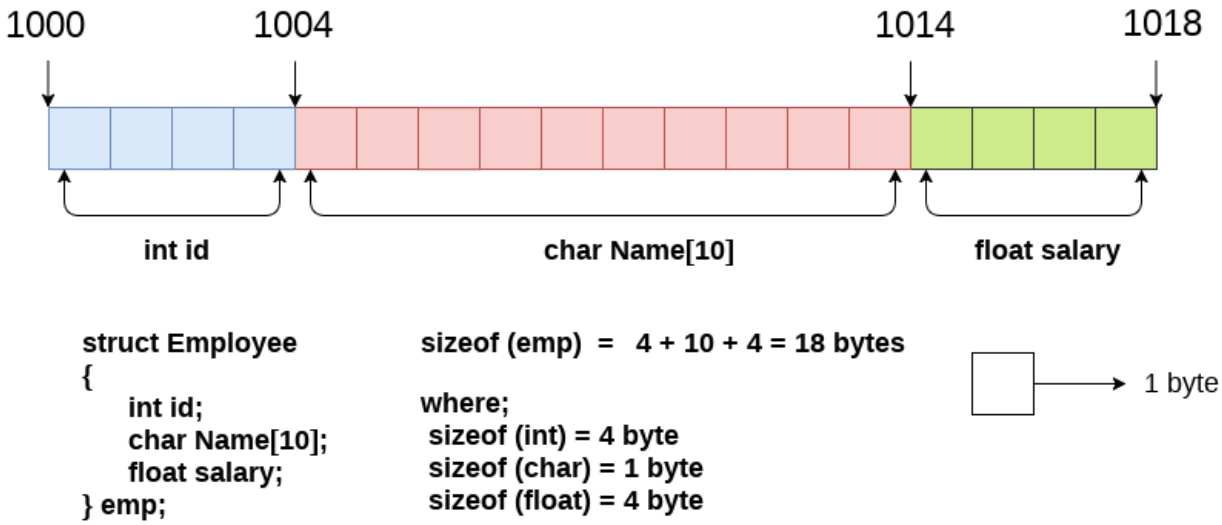
```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure –

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

```
struct employee
{   int id;
   char name[20];
   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.

Declaring Structure Variables

It is possible to declare variables of a **structure**, either along with structure definition or after the structure is defined. **Structure** variable declaration is similar to the declaration of any normal variable of any other datatype. Structure variables can be declared in following two ways:

1) Declaring Structure variables separately

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};

struct Student S1, S2;      //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
}S1, S2;
```

Here `S1` and `S2` are variables of structure `Student`. However, this approach is not much recommended. If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.

How to initialize structure members?

Structure members **cannot be** initialized with declaration. For example, the following C program fails in compilation.

```
struct Point
{
int x = 0; // COMPILER ERROR: cannot initialize members here
int y = 0; // COMPILER ERROR: cannot initialize members here
};
```

The reason for above error is simple, when a datatype is declared, no memory is allocated for it. Memory is allocated only when variables are created.

Structure members **can be** initialized using curly braces '{}'. For example, following is a valid initialization.

```
struct Point
{
int x, y;
};

int main()
{
// A valid initialization. member x gets value 0 and y
// gets value 1. The order of declaration is followed.
struct Point p1 = {0, 1};
}
```

Accessing Structure Members

Structure members can be accessed and assigned values in a number of ways. Structure members have no meaning individually without the structure. There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

By dot operator

In order to assign a value to any structure member, the member name must be linked with the **structure** variable using a dot . operator also called **period** or **member access** operator.

For example:

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

```
#include <string.h>
struct employee
{   int id;
    char name[50];
}e1;  //declaring e1 variable for structure
int main( )
{
   //store first employee information
   e1.id=101;
   strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```

OUTPUT

```
employee 1 id : 101
employee 1 name : Sonoo Jaiswal
#include<stdio.h>
#include<string.h>

struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};

int main()
{
    struct Student s1;

    /*
        s1 is a variable of Student type and
        age is a member of Student
    */
    s1.age = 18;
    /*
        using string function to add name
    */
    strcpy(s1.name, "Viraaj");
    /*
        displaying the stored values
    */
    printf("Name of Student 1: %s\n", s1.name);
    printf("Age of Student 1: %d\n", s1.age);

    return 0;
}
```

OUTPUT

Name of Student 1: Viraaj

Age of Student 1: 18

We can also use `scanf()` to give values to structure members through terminal.

```
scanf(" %s ", s1.name);
scanf(" %d ", &s1.age);
```

Designated Initialization allows structure members to be initialized in any order.

#include<stdio.h>

struct Point
{
int x, y, z;
};

int main()
{
// Examples of initialization using designated initialization
struct Point p1 = {.y = 0, .z = 1, .x = 2};
struct Point p2 = {.x = 20};

printf ("x = %d, y = %d, z = %d\n", p1.x, p1.y, p1.z);
printf ("x = %d", p2.x);
return 0;
}

OUTPUT

```
x = 2, y = 0, z = 1
x = 20
#include <stdio.h>
#include <string.h>

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */
```

```
   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);

   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 subject : %s\n", Book2.subject);
   printf( "Book 2 book_id : %d\n", Book2.book_id);

   return 0;
}
```

OUTPUT

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

Array of Structure

We can also declare an array of **structure** variables. in which each element of the array will represent a **structure** variable. **Example :** `struct employee emp[5];`

The below program defines an array `emp` of size 5. Each element of the array `emp` is of type `Employee`.

```
#include<stdio.h>

struct Employee
{
   char ename[10];
   int sal;
};
```

```
void main()
{
struct Employee emp[5];
int i, j;
for(i = 0; i < 3; i++)
   {
      printf("\nEnter %dst Employee record:\n", i+1);
      printf("\nEmployee name:\t");
      scanf("%s", emp[i].ename);
      printf("\nEnter Salary:\t");
      scanf("%d", &emp[i].sal);
   }
   printf("\nDisplaying Employee record:\n");
   for(i = 0; i < 3; i++)
   {
      printf("\nEmployee name is %s", emp[i].ename);
      printf("\nSlary is %d", emp[i].sal);
   }
}
```

OUTPUT

```
Enter 1st Employee record:
Employee name:  Hari
Enter Salary:   20000
Enter 2st Employee record:
Employee name:  Shyam
Enter Salary:   25000
Enter 3st Employee record:
Employee name:  Ram
Enter Salary:   30000
Displaying Employee record:
Employee name is Hari
Slary is 20000
Employee name is Shyam
Slary is 25000
Employee name is Ram
Slary is 30000
```

Nested Structures

Nesting of structures, is also permitted in C language. Nested structures means, that one structure has another stucture as member variable. 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.

Example:

```
#include<stdio.h>
struct address
{
char city[20];
int pin;
char phone[14];
};
struct employee
{
char name[20];
struct address add;
};
void main ()
{
struct employee emp;
printf("Enter employee information?\n");
scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
printf("Printing the employee information....\n");
printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,emp.add.pin,em
p.add.phone);
}
```

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

By 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.

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

By 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;
   }doj;
}emp1;
```

Accessing Nested Structure

We can access the member of the nested structure by Outer_Structure.Nested_Structure.member as given below:

```
e1.doj.dd
e1.doj.mm
e1.doj.yyyy
```

Example

```
#include <stdio.h>
#include <string.h>
struct Employee
{
  int id;
  char name[20];
  struct Date
   {
     int dd;
     int mm;
     int yyyy;
   }doj;
}e1;
int main( )
{
  //storing employee information
  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.y
yyy);
  return 0;
}
```

OUTPUT

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

Structure as Function Arguments

We can pass a structure as a function argument just like we pass any other variable or an array as a function argument.

Example

```
#include<stdio.h>

struct Student
{
    char name[10];
    int roll;
};

void show(struct Student st);

void main()
```

```
{
    struct Student std;
    printf("\nEnter Student record:\n");
    printf("\nStudent name:\t");
    scanf("%s", std.name);
    printf("\nEnter Student rollno.:\t");
    scanf("%d", &std.roll);
    show(std);
}

void show(struct Student st)
{
    printf("\nstudent name is %s", st.name);
    printf("\nroll is %d", st.roll);
}
```

OUTPUT

Enter Student record:
Student name: Hari
Enter Student rollno.: 1101
student name is Hari
roll is 1101

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

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

/* function declaration */
void printBook( struct Books book );

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   printBook( Book1 );
```

```
   /* Print Book2 info */
   printBook( Book2 );

   return 0;
}

void printBook( struct Books book ) {

   printf( "Book title : %s\n", book.title);
   printf( "Book author : %s\n", book.author);
   printf( "Book subject : %s\n", book.subject);
   printf( "Book book_id : %d\n", book.book_id);
}
```

OUTPUT

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Structure pointer operator

We can define pointers to structures in the same way as you define pointer to any other variable. Like primitive types, we can have pointer to a structure. If we have a pointer to structure, members are accessed using arrow (->) operator.

Example

```
struct Books *struct_pointer;
```

Now, we can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the '&'; operator before the structure's name as follows –

```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the → operator as follows –

```
struct_pointer->title;
```

Let us re-write the earlier discussed example using structure pointer.

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

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

/* function declaration */
```

```
void printBook( struct Books *book );
int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info by passing address of Book1 */
   printBook( &Book1 );

   /* print Book2 info by passing address of Book2 */
   printBook( &Book2 );

   return 0;
}

void printBook( struct Books *book ) {

   printf( "Book title : %s\n", book->title);
   printf( "Book author : %s\n", book->author);
   printf( "Book subject : %s\n", book->subject);
   printf( "Book book_id : %d\n", book->book_id);
}
```

OUTPUT

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```