## 2.9. Structures:

Arrays allows 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 allows combining data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. you might want to track the following attributes about each book.

* Title
* Author
* Subject
* Book ID

### Defining a Structure:

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;
  .....
};
```

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.

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

Accessing Structure Members:

To access any member of a structure, we use the member access operator (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword struct to define variables of structure type.

The following example shows how to use a structure in a program -

```
#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;
 /* Book2 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 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 ("Book2 subject : %s\n", Book2.subject);
printf ("Book2 book_id : %d\n", Book2.book_id);
return 0;
}
```

When the above code is compiled and executed, it produces the following result -

```
Book 1 title : C programming Book
1 author : Nuha Ali
Book 1 subject : C programming Tutorial Book 1
book_id : 6495407
Book2 title : Telecom Billing Book 2
author : Zara Ali
Book 2 subject : Telecom Billing Tutorial Book 2
book_id : 6495700
```

Example:

```
#include <stdio.h>
#include <string.h>
// Create struct with person1 variable
struct person
{
  char name[50];
  int citNo;
  float salary;
} person1;
int main()
{     // assign value to name of person1
  strcpy(person1.name, "George Orwell");
  // assign values to other person1 variables
  person1.citNo = 1984;
  person1.salary = 2500;
  // print struct variables
  printf("Name : %s \n", person1.name);
  printf("Citizenship NO : %d \n", person1.citNo);
  printf("Salary : %f", person1.salary);
  return 0;
}
```

Output

```
Name : George Orwell
Citizenship NO : 1984
Salary : 2500.00
```

## 2.10. Nested Structures:

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.

```
#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 \n city : %s \n pincode : %d \n Phone:%s",
        emp.name, emp.add.city, emp.add.pin, emp.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

1). Separate structure

Here, we create two structures, but the ~~department~~ 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;
} empl;
```

2). Embedded Structure

The embedded structure enables us to declare the structure inside the structure. Hence, it requires less line of codes but it cannot 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;
} empl;
```