STRUCTURES, UNIONS AND FILE HANDLING IN C

Structure: Declaration, Definition-Array of Structures - Pointer to Structure –Nested Structures- Union: Defining union, Accessing union members. Files: File Management functions, Random access in file- Working with Text Files and Binary Files.

5.1 STRUCTURE: DECLARATION, DEFINITION

A **structure** in C is a derived or user-defined data type. We use the keyword **struct** to define a custom data type that groups together the elements of different types. The difference between an array and a structure is that an array is a homogenous collection of similar types, whereas a structure can have elements of different types stored adjacently and identified by a name.

We are often required to work with values of different data types having certain relationships among them. For example, a **book** is described by its **title** (string), **author** (string), **price** (double), **number of pages** (integer), etc. Instead of using four different variables, these values can be stored in a single **struct** variable.

a) Declare (Create) a Structure

We can create (declare) a structure by using the **"struct"** keyword followed by the structure_tag (structure name) and declare all of the members of the structure inside the curly braces along with their data types. To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member.

Syntax of Structure Declaration

The format (syntax) to declare a structure 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**.

Example

In the following example we are declaring a structure for Book to store the details of a Book

```
struct book{
   char  title[50];
   char  author[50];
   double price;
   int   pages;
} book1;
```

Here, we declared the structure variable **book1** at the end of the structure definition. However, you can do it separately in a different statement.

b) Structure Variable Declaration

To access and manipulate the members of the structure, you need to declare its variable first. To declare a structure variable, write the structure name along with the **"struct"** keyword followed by the name of the structure variable. This structure variable will be used to access and manipulate the structure members.

Example

The following statement demonstrates how to declare (create) a structure variable

```
struct book book1;
```

Usually, a structure is declared before the first function is defined in the program, after the **include** statements. That way, the derived type can be used for declaring its variable inside any function.

c) Structure Initialization

The **initialization** of a struct variable is done by placing the value of each element inside curly brackets.

Example

The following statement demonstrates the initialization of structure

```
struct book book1 = {"Learn C", "Dennis Ritchie", 675.50, 325};
```

d) Accessing the Structure Members

To access the members of a structure, first, you need to declare a structure variable and then use the **dot (.) operator** along with the structure variable.

Example 1

The four elements of the struct variable book1 are accessed with the **dot (.) operator**. Hence, "book1.title" refers to the title element, "book1.author" is the author name, "book1.price" is the price, "book1.pages" is the fourth element (number of pages).

Take a look at the following example −

```c
#include <stdio.h>

struct book{
   char title[10];
   char author[20];
   double price;
   int pages;
};

int main(){
   struct book book1 = {"Learn C", "Dennis Ritchie", 675.50, 325};

   printf("Title:  %s \n", book1.title);
   printf("Author: %s \n", book1.author);
   printf("Price:  %lf\n", book1.price);
   printf("Pages:  %d \n", book1.pages);
   printf("Size of book struct: %d", sizeof(struct book));
   return 0;
}
```

Output

Title: Learn C

Author: Dennis Ritchie

Price: 675.500000

Pages: 325

Size of book struct: 48

Example 2

In the above program, we will make a small modification. Here, we will put the **type definition** and the **variable declaration** together, like this −

```c
struct book{
    char title[10];
    char author[20];
    double price;
    int pages;
} book1;
```

Note that if you a declare a struct variable in this way, then you cannot initialize it with curly brackets. Instead, the elements need to be assigned individually.

```c
#include <string.h>

struct book{
    char title[10];
    char author[20];
    double price;
    int pages;
} book1;

int main(){
    strcpy(book1.title, "Learn C");
    strcpy(book1.author, "Dennis Ritchie");
    book1.price = 675.50;
    book1.pages = 325;

    printf("Title: %s \n", book1.title);
    printf("Author: %s \n", book1.author);
    printf("Price: %lf \n", book1.price);
    printf("Pages: %d \n", book1.pages);
    return 0;
}
```

Output

Title: Learn C

Author: Dennis Ritchie

Price: 675.500000

Pages: 325