5.2 ARRAY OF STRUCTURES IN C

In C programming, the **struct** keyword is used to define a derived data type. Once defined, you can declare an array of struct variables, just like an array of **int**, **float** or **char** types is declared. An array of structures has a number of use-cases such as in storing records similar to a database table where you have each row with different data types.

Usually, a struct type is defined at the beginning of the code so that its type can be used inside any of the functions. You can declare an array of structures and later on fill data in it or you can initialize it at the time of declaration itself.

Initializing a Struct Array

Let us define a struct type called **book** as follows −

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

During the program, you can declare an array and initialize it by giving the values of each element inside curly brackets. Each element in the struct array is a struct value itself. Hence, we have the nested curly brackets as shown below −

```
struct book b[3] = {
   {"Learn C", 650.50, 325},
   {"C Pointers", 175, 225},
   {"C Pearls", 250, 250}
};
```

How does the compiler allocate memory for this array? Since we have an array of three elements, of **struct** whose size is 32 bytes, the array occupies "32 x 3" bytes. Each block of 32 bytes will accommodate a "title", "price" and "pages" element.

L	E	A	R	N		C				**675.50**	**325**
C	P	O	I	N	T	E	R	S		**175**	**225**
C		P	E	A	R	L	S			**250**	**250**

a) Declaring a Struct Array

You can also declare an empty struct array. Afterwards, you can either read the data in it with scanf() statements or assign value to each element as shown below −

```
struct book b[3];
strcpy(b[0].title, "Learn C");
b[0].price = 650.50;
b[0].pages=325;

strcpy(b[1].title, "C Pointers");
b[1].price = 175;
b[1].pages=225;

strcpy(b[2].title, "C Pearls");
b[2].price = 250;250
b[2].pages=325;
```

b) Reading a Struct Array

We can also accept data from the user to fill the array.

Example 1

In the following code, a **for** loop is used to accept inputs for the "title", "price" and "pages" elements of each struct element of the array.

```c
#include <stdio.h>

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

int main(){

   struct book b[3];

   strcpy(b[0].title, "Learn C");
   b[0].price = 650.50;
   b[0].pages = 325;

   strcpy(b[1].title, "C Pointers");
   b[1].price = 175;
   b[1].pages = 225;

strcpy(b[2].title, "C Pearls");
   b[2].price = 250;
   b[2].pages = 325;
   printf("\nList of Books:\n");
   for (int i = 0; i < 3; i++){
      printf("Title: %s \tPrice: %7.2lf \tPages: %d\n", b[i].title, b[i].price, b[i].pages);
   }

   return 0;
}
```

Output

List of Books:

Title: Learn C Price: 650.50 Pages: 325

Title: C Pointers Price: 175.00 Pages: 225

Title: C Pearls Price: 250.00 Pages: 325

c) Sorting a Struct Array

Let us take another example of struct array. Here, we will have the array of "book" struct type sorted in ascending order of the price by implementing bubble sort technique.

Note: The elements of one struct variable can be directly assigned to another struct variable by using the assignment operator.

Example

Take a look at the example

```c
#include <stdio.h>
struct book{
   char title[15];
   double price;
   int pages;
};

int main(){

   struct book b[3] = {
      {"Learn C", 650.50, 325},
      {"C Pointers", 175, 225},
      {"C Pearls", 250, 250}
   };

   int i, j;
   struct book temp;
```

```c
for(i = 0; i < 2; i++){
   for(j = i; j < 3; j++){
      if (b[i].price > b[j].price){
         temp = b[i];
         b[i] = b[j];
         b[j] = temp;
      }
   }
}

printf("\nList of Books in Ascending Order of Price:\n");

for (i = 0; i < 3; i++){
```

```
    printf("Title: %s \tPrice: %7.2lf \tPages: %d\n", b[i].title, b[i].price, b[i].pages);
  }
  return 0;
}
```

Output

List of Books in Ascending Order of Price:

Title: C Pointers Price: 175.00 Pages: 225

Title: C Pearls Price: 250.00 Pages: 250

Title: Learn C Price: 650.50 Pages: 325