Data Structures in C

C arrays allow you to define type of variables that can hold several data items of the same kind but **structure** is another user defined data type available in C programming, which allows you to combine 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 for your program. The format of the struct statement is this:

```
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:

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

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 **struct** keyword to define variables of structure type. Following is the example to explain usage of structure:

```
#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;
}
```

Structures as Function Arguments

You can **pass a structure as a function argument** in very similar way as you pass any other
variable or pointer. You would access structure variables in the similar way as you have accessed
in the above example:

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

struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
```

```c
    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);
}
```

Pointers to Structures

You can define pointers to structures in very similar way as you define pointer to any other variable as follows:

```c
struct Books *struct_pointer;
```

Now, you 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:

```c
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 above example using **structure pointer**, hope this will be easy for you to understand the concept:

```c
#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);
}
```

http://www.tutorialspoint.com/cprogramming/c_structures.htm