# Structures in C

# What is a structure

A *struct* (or structure) is a collection of variables (can be of different types) under a single name

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.

# What is the purpose

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

Use the **struct** statement.

The struct statement defines a new data type

The format of the struct statement is as follows −

```
struct [structure tag] {

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

# How to declare and define

The structure tag is optional
Each member definition is a normal variable definition,
Such as int i;
or float f;

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

# Accessing Structure Members

Use - **member access operator (.)**.

int i;

struct Books Book1;

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

/* Declare Book2 of type Book */

# Accessing Structure Members

Book1.book_id = 6495407;

# A full example

```
#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

# Structures as Function Arguments

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

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

};

void printBook_x( 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);
}
```

```
void printBook_y( 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);
}
```

```
/* function declaration */
void printBook_x( struct Books book );
void printBook_x( 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;

printBook_x(Book1);                    // it works through call by value
printBook_y(&Book1);                   // when it works through call by address
```

```
   /* 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;
}
```

# 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

# Pointers to structures

```
struct Books Book1;

struct Books *struct_pointer;

struct_pointer = &Book1;

struct_pointer->title;

Book1.title;
```

# Pointers to structures

```
struct Books Book[2];

struct Books *struct_pointer;

struct_pointer = &Book[0];

struct_pointer->title;

Book[0].title;
```

struct_pointer= struct_pointer+1;

```
#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);
}
```

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

# Nested Structures

```
struct complex
{
 int imag;
 float real;
};

struct number
{
   struct complex comp;

} num1, num2;
```

Imagine a number will have three components –

fraction_part + i. imaginary_component

i = sqrt(-1).

```
struct organization{

    struct employ[100];

    struct students[100];


    struct Books[5];
}
```

```
struct organization iitbbs;

struct organization *org_ptr;

org_ptr = & iitbbs;

org_ptr->book.id ….
```

# Nested structure

Set imag of num2 variable to 11.

struct number num1, num2;

num1.integers = 10;
num1.comp.imag = 10;
num1.comp.real = 10;

- Complex numbers have two parts – real and imaginary.
- Separate real and imaginary parts
- Add, Subtract, Mul, Div

```
struct complex add(struct complex *c1, struct
complex *c2 ){

        struct complex temp;

        temp.real = C1->real + C2->real
        temp.imag = C1->imag + C2->image

        return temp;
}
```

```
struct complex *add(struct complex *c1, struct complex *c2 ){

        struct complex *temp = (struct complex *)malloc(sizeof(struct complex));

        temp->real = C1->real + C2->real;
        temp->imag = C1->imag + C2->image;

        return temp;
}
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {

int data;
int key;
struct node *next;

};

struct node *head = NULL;

struct node *current = NULL;
```

```
//display the list

void printList()
{
struct node *ptr = head;

printf("\n[ ");

//start from the beginning

while(ptr != NULL) {
        printf("(%d,%d) ",ptr->key,ptr->data);
        ptr = ptr->next; } printf(" ]");
}
```

```
//insert link at the first location

void insertFirst(int key, int data)
{
        //create a link

        struct node *link = (struct node*)      malloc(sizeof(struct
node));

        link->key = key; link->data = data;
        //point it to old first node
        link->next = head;
        //point first to new first node
        head = link;
}
```

```
//delete first item
struct node* deleteFirst()
{
        //save reference to first link
        struct node *tempLink = head;
        //mark next to first link as first
        head = head->next;
        //return the deleted link
        return tempLink;
}
```

```
//is list empty
bool isEmpty()
{
        return head == NULL;
}
int length() {

        int length = 0;
        struct node *current;
        for(current = head; current != NULL; current =
current->next)
                { length++; }
return length;
}
```

```
//find a link with given key

struct node* find(int key)
{

//start from the first link
struct node* current = head;

//if list is empty
if(head == NULL) {
        return NULL;
}

//navigate through list
while(current->key != key) {
//if it is last node
        if(current->next == NULL)
        { return NULL; }
        else {
        //go to next link
        current = current->next;
        }
} //if data found, return the current
Link
return current; }
```

```
//delete a link with given key
struct node* delete(int key)
{ //start from the first link

struct node* current = head;
struct node* previous = NULL;
//if list is empty if(head == NULL) {
        return NULL;
}
//navigate through list

while(current->key != key) { //if it is last node
        if(current->next == NULL)
                { return NULL; }
        else { //store reference to current link
                previous = current;
                //move to next link
                current = current->next;
        }
}
```

```
    //found a match, update the link
    if(current == head) {
        //change first to point to next link
        head = head->next;
    } else {

        //bypass the current link
        previous->next = current->next;
    }
    return current;
}
```

```
void sort()

{
int i, j, k, tempKey,
tempData;

struct node *current;
struct node *next;
int size = length();
k = size ;
for ( i = 0 ; i < size - 1 ; i++,
k-- )
     {
     }
}
```

```
current = head;
next = head->next;
for ( j = 1 ; j < k ; j++ )
{
          if ( current->data > next->data )
                    { tempData = current->data;
                    current->data = next->data;
                    next->data = tempData;
                    tempKey = current->key;
                    current->key = next->key;
                    next->key = tempKey;
          }
          current = current->next; next =
next->next;
}
```

```
void reverse(struct node** head_ref)
{

struct node* prev = NULL;
struct node* current = *head_ref;
struct node* next;
while (current != NULL)
{
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
}

*head_ref = prev;

}
```

```
void main() {

insertFirst(1,10);
insertFirst(2,20);
insertFirst(3,30);
insertFirst(4,1);
insertFirst(5,40);
insertFirst(6,56);

printf("Original List: ");
//print list
printList();
```

```
while(!isEmpty())
{

struct node *temp =
deleteFirst();
printf("\nDeleted value:");
printf("(%d,%d) ",temp-
>key,temp->data);
}

printf("\nList after deleting all
items: ");
printList();
```

```
insertFirst(1,10);

insertFirst(2,20);

insertFirst(3,30);

insertFirst(4,1);

insertFirst(5,40);

insertFirst(6,56);

printf("\nRestored List: ");

printList();

printf("\n");
```

```
struct node *foundLink = find(4);

if(foundLink != NULL) {
      printf("Element found: ");
      printf("(%d,%d) ",
      foundLink->key,foundLink->data);
      printf("\n");
}  else {
            printf("Element not found.");
}

delete(4);

printf("List after deleting an item: ");
printList();
printf("\n");
```

```
foundLink = find(4);
if(foundLink != NULL) {
        printf("Element found: ");
        printf("(%d,%d) ",
        foundLink->key,foundLink-
>data);          printf("\n");
} else {

printf("Element not found.");


}

printf("\n");
sort();
printf("List after sorting the data:
");
printList();
```

```
reverse(&head);

printf("\nList after
reversing the data: ");

printList(); }
```

Original List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10) ]

Deleted value:(6,56)

Deleted value:(5,40)

Deleted value:(4,1)

Deleted value:(3,30)

Deleted value:(2,20)

Deleted value:(1,10)

List after deleting all items:
[ ]

Restored List:

[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10) ]

Element found: (4,1)

List after deleting an item:

[ (6,56) (5,40) (3,30) (2,20) (1,10) ]

Element not found.

List after sorting the data:

[ (1,10) (2,20) (3,30) (5,40) (6,56) ]

List after reversing the data:

[ (6,56) (5,40) (3,30) (2,20) (1,10) ]