# Andhra Pradesh State Skill Development Corporation

Skill AP
APSSDC



# Programming in C

# Structures

# Structures

## What is Structure

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information.

The struct keyword is used to define the structure. Let's see the syntax to define the structure in c.

## Syntax:

```
struct structure_name
{
  data_type member1;
  data_type member2;
  .
  .
  data_type memeberN;
};
```

## Use structure

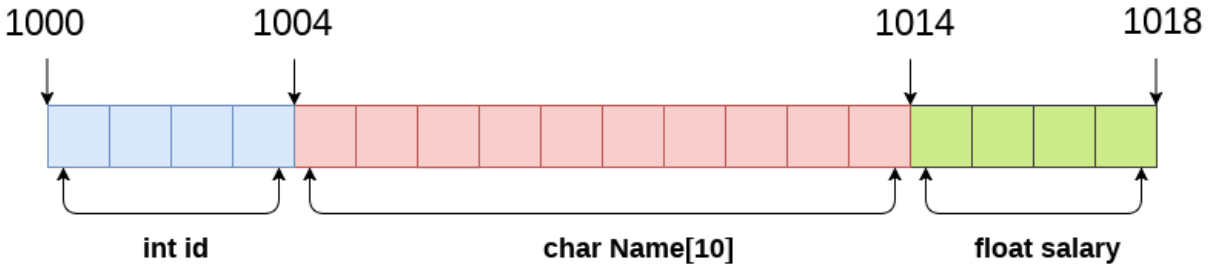
In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity Student may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

- Construct individual arrays for storing names, roll numbers, and marks.
- Use a special data structure to store the collection of different data types.

Example:

```
struct employee
{   int id;
    char name[20];
    float salary;
};
```

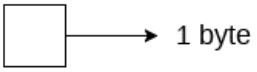
Memory allocation for above example:



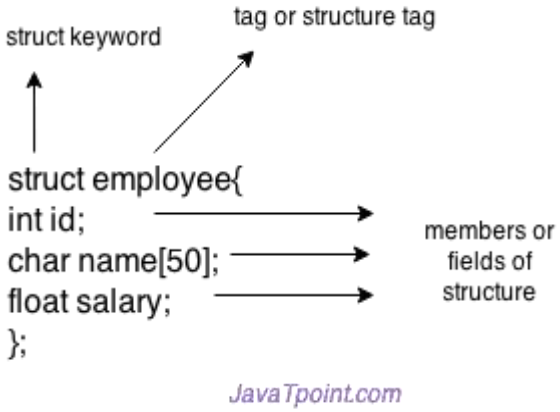

```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;
sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte




Here, struct is the keyword; employee is the name of the structure; id, name, and salary are the members or fields of the structure. Let's understand it by the diagram given below:




Example for data storing of single person :

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
}e1;  //declaring e1 variable for structure
int main( )
{
   //store first employee information
   e1.id=101;
   strcpy(e1.name, "Jyothi");//copying string into char array
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```

Output:

```
employee 1 id : 101
employee 1 name : Jyothi
```

Another example of the structure  store many employees information:

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;  //declaring e1 and e2 variables for structure
int main( )
{
   //store first employee information
   e1.id=101;
   strcpy(e1.name, "Suresh");//copying string into char array
   e1.salary=56000;

  //store second employee information
   e2.id=102;
   strcpy(e2.name, "Ramesh");
   e2.salary=126000;

   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
   printf( "employee 1 salary : %f\n", e1.salary);

   //printing second employee information
   printf( "employee 2 id : %d\n", e2.id);
   printf( "employee 2 name : %s\n", e2.name);
   printf( "employee 2 salary : %f\n", e2.salary);
   return 0;
}
```

**Output:**

```
employee 1 id : 101
employee 1 name : Suresh
employee 1 salary : 56000.000000
employee 2 id : 102
employee 2 name : Ramesh
employee 2 salary : 126000.000000
```

## Structures as Function Arguments:

You can pass a structure as a function argument in the same way as you pass any other variable or pointer.

Example:

```
#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, "abc");
   strcpy( Book1.subject, "C Programming ");
   Book1.book_id = 123;

   /* book 2 specification */
   strcpy( Book2.title, "Java");
   strcpy( Book2.author, "xyz");
   strcpy( Book2.subject, "Java");
   Book2.book_id = 456;

   /* 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);
}
```

**Output:**

```
Book title : C Programming
Book author : abc
Book subject : C Programming
Book book_id : 123
Book title : Java
Book author : xyz
Book subject : Java
Book book_id : 456
```

## Pointers to Structures

You can define pointers to structures in the same way as you define pointer to any other variable −

```
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 −

```
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;
```

Example:

```
#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, "abc");
   strcpy( Book1.subject, "C Programming ");
```

```
   Book1.book_id = 123;

   /* book 2 specification */
   strcpy( Book2.title, "java");
   strcpy( Book2.author, "xyz");
   strcpy( Book2.subject, "java");
   Book2.book_id = 456;

   /* 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);
}
```

**Output:**

```
Book title : C Programming
Book author : abc
Book subject : C Programming
Book book_id : 123
Book title : java
Book author :xyz
Book subject : java
Book book_id : 456
```

**** THE END ****