Structures in C

When we design a program, it is important to choose an optimal way to represent the data. In simple cases, scalar variables and arrays are sufficient. However, in practical programming context, you need a new form of variable that reflects the real world. For example, in your program, you may want to refer to an address that holds multiple fields including house number, street, zip code, state and country.

C provides a special kind of variable called structure. C structure allows you to wrap related variables that has different data types into a single variable. A structure can contain any valid data types such as int, char, float, array, pointer or even other structures. Each variable in the structure is called structure member.

Defining structure

To define a structure, we use the *struct* keyword. The following illustrates the syntax of the structure definition

```
struct struct_name{ structure_member };
```

The following example defines the address structure

```
struct address{
   unsigned int house_number;
   char street_name[50];
   int zip_code;
   char country[50];
 };
```

The address structure contains house_number as an unsigned int, street_name as a string, zip_code as an int and country as a string.

Declaring Structure Variables

The above example only defines a structure without creating any structure variable. There are two ways to declare a structure variable:

- Declare **structure variables** together with the structure definition

For example, to define *home_address* and *business_address* structures, we can use the following

```
struct address{
   unsigned int house_number;
   char street_name[50];
   int zip_code;
   char country[50];
 } home_address, business_address;
```

- Declare the **structure variable** after define the structure

For example, to define *home_address* and *business_address* as a structure variables after we define the structure we may use the following

```
struct address home_address, business_address;
```

Accessing structure member

To access a structure member, we can use the dot operator (.) between structure name and the member as in following example

```
struct address billing_addr;
billing_addr. zip_code = 9660011;
```

Pointer to a Structure in C

We have already learned that a pointer is a variable which points to the address of another variable of any data type like **int**, **char**, **float** etc. Similarly, we can have a pointer to structure, where a pointer variable can point to the address of a structure variable. Here is how we can declare a pointer to a structure variable.

```
struct dog
{
    char name[10];
    char breed[10];
    int age;
    char color[10];
};

struct dog spike;

// declaring a pointer to a structure of type struct dog
struct dog *ptr_dog;
```

This declares a pointer **ptr_dog** that can store the address of the variable of type struct dog. We can now assign the address of the variable **spike** to **ptr_dog** using & operator.

```
ptr_dog = &spike;
```

Now **ptr_dog** points to the structure variable **spike**.

Accessing members using Pointer

There are two ways of accessing members of structure using pointer:

1- Using indirection (*) operator and dot (.) operator.

In the previous example ptr_dog points to the structure variable spike, so by dereferencing it we will get the contents of the spike.

This means spike and *ptr_dog are functionally equivalent. To access a member of structure write *ptr_dog followed by a dot(.) operator, followed by the name of the member. For example:

```
(*ptr_dog).name    // refers to the name of dog
(*ptr_dog).breed   // refers to the breed of dog
```

2- Using arrow (->) operator

The above method of accessing members of the structure using pointers is slightly confusing and less readable, that's why C provides another way to access members using the arrow (->) operator. To access members using arrow (->) operator write pointer variable followed by -> operator, followed by name of the member. For example:

```
ptr_dog -> name    // refers to the name of dog
ptr_dog -> breed   // refers to the breed of dog
```

- Here we know that the name of the array (ptr_dog->name) is a constant pointer and points to the 0th element of the array. So we can't assign a new string to it using assignment operator (=), that's why strcpy() function is used.

```
strcpy( ptr_dog->name, "new_name");
```

Investigate the following example

```
#include<stdio.h>
#include <string.h>
struct dog
{
    char name[10];
    char breed[10];
    int age;
    char color[10];
};
int main()
{
    struct dog my_dog = {"tyke", "Bulldog", 5, "white"};
    struct dog *ptr_dog;
    ptr_dog = &my_dog;
    printf("Dog's name: %s\n", ptr_dog->name);
    printf("Dog's breed: %s\n", ptr_dog->breed);
    printf("Dog's age: %d\n", ptr_dog->age);
    printf("Dog's color: %s\n", ptr_dog->color);
    // changing the name of dog from tyke to jack
    strcpy(ptr_dog->name, "jack");
    // increasing age of dog by 1 year
    ptr_dog->age++;
    printf("Dog's new name is: %s\n", ptr_dog->name);
    printf("Dog's age is: %d\n", ptr_dog->age);
    // signal to operating system program ran fine
    return 0;
}
```

Investigate the following example

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

struct student
{
            int id;
            char name[20];
            float percentage;
} record;

int main()
{

      record.id=1;
      strcpy(record.name, "Lena");
      record.percentage = 86.5;

      printf(" Id is: %d \n", record.id);
      printf(" Name is: %s \n", record.name);
      printf(" Percentage is: %f \n", record.percentage);
  return 0;
}
```

Array of Structures

C Structure is collection of different datatypes which are grouped together. Whereas, array of structures is collection of structures. This is also called as structure array in C

Investigate the following example

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

struct student
{
     int id;
     char name[30];
     float percentage;
};
int main()
{
     int i;
     struct student record[3];

     // 1st student's record
     record[0].id=1;
     strcpy(record[0].name, "Raju");
     record[0].percentage = 86.5;

     // 2nd student's record
     record[1].id=2;
     strcpy(record[1].name, "Surendren");
     record[1].percentage = 90.5;

     // 3rd student's record
     record[2].id=3;
     strcpy(record[2].name, "Thiyagu");
     record[2].percentage = 81.5;

 for(i=0; i<3; i++)
 {
   printf(" Records of STUDENT : %d \n", i+1);
   printf(" Id is: %d \n", record[i].id);
   printf(" Name is: %s \n", record[i].name);
   printf(" Percentage is: %f\n\n",record[i].percentage);
 }
     return 0;
 }
```

Passing Structure to Function in C

It can be done in 3 Possible ways

1- *Passing Structure to Function by value*

The whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values.

Investigate the following example

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float percentage;
};
void func(struct student record);
int main()
{
     struct student john;
     john.id=1;
     strcpy(john.name, "John Jolyan");
     john.percentage = 86.5;
     func(john);
      return 0;
}
void func(struct student record)
{
     printf(" Id is: %d \n", record.id);
     printf(" Name is: %s \n", record.name);
     printf(" Percentage is: %f \n", record.percentage);
}
```

2- *Passing Structure to Function by Address*

In this procedure the whole structure is passed to another function by address. It means only the address of the structure is passed to another function.

Investigate the following example

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

struct student
{
    int id;
    char name[20];
    float percentage;
};

void func(struct student *record);

int main()
{
     struct student John;
     John.id=1;
     strcpy(John.name, " John Jolyan");
     John.percentage = 86.5;

     func(&John);
     return 0;
}

void func(struct student *record)
{
    printf(" Id is: %d \n", record->id);
    printf(" Name is: %s \n", record->name);
    printf(" Percentage is: %f \n", record->percentage);
}
```

3- *No need to pass a structure – Declare structure variable as global*

Structure variables also can be declared as global variables as we declare other variables in C. When a structure variable is declared as global, then it is visible to all the functions in a program.

Investigate the following example

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

struct student
{
            int id;
            char name[20];
            float percentage;
};
struct student record; // Global declaration of structure

void structure_demo();

int main()
{
      record.id=1;
      strcpy(record.name, "Raju");
      record.percentage = 86.5;

      structure_demo();

  return 0;
}

void structure_demo()
{
   printf(" Id is: %d \n", record.id);
   printf(" Name is: %s \n", record.name);
   printf(" Percentage is: %f \n", record.percentage);
}
```

Nested Structure

Nested structure in C is a structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.

Investigate the following example

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

struct college_details
{
    int college_id;
    char college_name[50];
};

struct student_details
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct college_details data;
}Ales;

int main()
{
 struct student_details Ales = {1, "Alis Noble", 90.5, 71145,"EMU"};
 printf(" Id is: %d \n", Ales.id);
 printf(" Name is: %s \n", Ales.name);
 printf(" Percentage is: %f \n\n", Ales.percentage);
 printf(" College Id is: %d \n",Ales.data.college_id);
 printf(" College Name is: %s \n",Ales.data.college_name);
 return 0;
}
```

Shorthand structure with *typedef* keyword

To make the code clear, you can use *typedef* keyword to create a synonym for a structure. By this definition we can omit the struct keyword

Investigate the following example

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

typedef struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
};
int main( ) {
   Books book;
   strcpy( book.title, "C Programming");
   strcpy( book.author, "Ruy");
   strcpy( book.subject, "C Programming Tutorial");
   book.book_id = 6495407;
   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);
   return 0;
}
```