C Structures

Structure is a user-defined datatype in C language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

For example: If I have to write a program to store Student information, which will have Student's name, age, branch, permanent address, father's name etc, which included string values, integer values etc, how can I use arrays for this problem, I will require something which can hold data of different types together.

In structure, data is stored in form of **records.**

Defining a structure

struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived data types.

Syntax:

```
struct [structure_tag]
{
 //member variable 1
```

```
//member variable 2
//member variable 3 ...
}
[structure_variables];
```

As you can see in the syntax above, we start with the struct keyword, then it's optional to provide your structure a name, we suggest you to give it a name, then inside the curly braces, we have to mention all the member variables, which are nothing but normal C language variables of different types like int, float, array etc.
After the closing curly brace, we can specify one or more structure variables, again this is optional.

Note: The closing curly brace in the structure type declaration must be followed by a semicolon(;).

Example of Structure

```
struct Student
 {
char name[25];
int age;
char branch[10];
// F for female and M for male
char gender;
};
```

Here struct Student declares a structure to hold the

details of a student which consists of 4 data fields, namely name, age, branch and gender. These fields are called **structure elements or members**.

Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. **Student** is the name of the structure and is called as the **structure tag.**

Declaring Structure Variables

It is possible to declare variables of a **structure**, either along with structure definition or after the structure is defined. **Structure** variable declaration is similar to the declaration of any normal variable of any other datatype. Structure variables can be declared in following two ways:

1) Declaring Structure variables separately

```
struct Student
{ char name[25];
int age;
char branch[10];
 //F for female and M for male
char gender; };
struct Student S1, S2;
 //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

```
struct Student
{
char name[25];
int age; char branch[10];
//F for female and M for male
char gender;
 }
S1, S2;
```

Here S1 and S2 are variables of structure Student. However this approach is not much recommended.

Accessing Structure Members

Structure members can be accessed and assigned values in a number of ways. Structure members have no meaning individually without the structure. In order to assign a value to any structure member, the member name must be linked with the **structure** variable using a dot . operator also called **period** or **member access** operator.

For example

```
#include<stdio.h>
#include<string.h>
struct Student
{
char name[25];
int age;
char branch[10];
//F for female and M for male
```

```
char gender;
};
int main()
{
struct Student s1;
/* s1 is a variable of Student type and age is a member
of Student */
s1.age = 18;
/* using string function to add name */
strcpy(s1.name, "Viraaj");
/* displaying the stored values */
printf("Name of Student 1: %s\n", s1.name);
printf("Age of Student 1: %d\n", s1.age);
return 0;
 }
```

We can also use scanf() to give values to structure members through terminal.

```
scanf(" %s ", s1.name);
scanf(" %d ", &s1.age);
```

Structure Initialization

Like a variable of any other datatype, structure variable can also be initialized at compile time.

```
struct Patient { float height; int weight; int age; }; struct
Patient p1 = { 180.75 , 73, 23 };
//initialization
```

or,

```
struct Patient p1;
p1.height = 180.75;
//initialization of each member separately p1.weight =
73;
p1.age = 23;
```

Last modified: 2:59 pm