Structure

C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.
If you want to access structure members in C, structure variable should be declared.
Many structure variables can be declared for same structure and memory will be allocated for each separately.
It is a best practice to initialize a structure to null while declaring, if we don’t assign any values to structure members.

DIFFERENCE BETWEEN C VARIABLE, C ARRAY AND C STRUCTURE:
A normal C variable can hold only one data of one data type at a time.
An array can hold group of data of same data type.
A structure can hold group of data of different data types and Data types can be int, char, float, double and long double etc.

C Structure:

Syntax	struct student
	{
	int a;
	char b[10];
	}

Example	a = 10;
	b = “Hello”;

C Variable:

int	Syntax: int a;
	Example: a = 20;

char	Syntax: char b; Example: b=’Z’;

C Array:

int	**Syntax:** int a[3];
	Example:
	a[0] = 10;
	a[1] = 20;
	a[2] = 30;
	a[3] = ‘\0’;

char	**Syntax:** char b[10];
	Example:
	b=”Hello”;

BELOW TABLE EXPLAINS FOLLOWING CONCEPTS IN C STRUCTURE.
How to declare a C structure?
How to initialize a C structure?
How to access the members of a C structure?

Using normal variable	**Using pointer variable**

Syntax:
struct tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
	Syntax:
	struct tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};

Example:
struct student
{
int mark;
char name[10];
float average;
};
	Example:
	struct student
	{
	int mark;
	char name[10];
	float average;
	};

Declaring structure using normal variable: struct student report;	**Declaring structure using pointer variable:** struct student *report, rep;

Initializing structure using normal variable:
struct student report = {100, “Mani”, 99.5};
	Initializing structure using pointer variable:
	struct student rep = {100, “Mani”, 99.5};
	report = &rep;

Accessing structure members using normal variable:
report.mark;
report.name;
report.average;
	Accessing structure members using pointer variable:
	report -> mark;
	report -> name;
	report -> average;

EXAMPLE PROGRAM FOR C STRUCTURE:
This program is used to store and access “id, name and percentage” for one student.

```
#include <stdio.h>
#include <string.h>
struct student
{
        int id;
        char name[20];
        float percentage;
};

int main()
{
        struct student record = {0}; //Initializing to null

        record.id=1;
        strcpy(record.name, "Raju");
        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;
}
```

OUTPUT:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

EXAMPLE PROGRAM – ANOTHER WAY OF DECLARING C STRUCTURE:

In this program, structure variable “record” is declared while declaring structure itself. In above structure example program, structure variable “struct student record” is declared inside main function which is after declaring structure.

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

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

int main()
{

        record.id=1;
        strcpy(record.name, "Raju");
        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;
 }
```

OUTPUT:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

C STRUCTURE DECLARATION IN SEPARATE HEADER FILE:
In above structure programs, C structure is declared in main source file. Instead of declaring C structure in main source file, we can have this structure declaration in another file called “header file” and we can include that header file in main source file as shown below.
HEADER FILE NAME – STRUCTURE.H
Before compiling and executing below C program, create a file named “structure.h” and declare the below structure.

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

MAIN FILE NAME – STRUCTURE.C:
In this program, above created header file is included in “structure.c” source file as #include “Structure.h”. So, the structure declared in “structure.h” file can be used in “structure.c” source file.

```
 // File name - structure.c
 #include <stdio.h>
 #include <string.h>
 #include "structure.h"   /* header file where C structure is  declared */
  int main()
 {
     record.id=1;
    strcpy(record.name, "Raju");
    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;
 }
```

OUTPUT:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

USES OF STRUCTURES IN C:
C Structures can be used to store huge data. Structures act as a database.
C Structures can be used to send data to the printer.
C Structures can interact with keyboard and mouse to store the data.
C Structures can be used in drawing and floppy formatting.
C Structures can be used to clear output screen contents.
C Structures can be used to check computer's memory size etc.

Another way to declare the structure variables-
Structure variable declaration
When a structure is defined, it creates a user-defined type but, no storage or memory is allocated.
For the above structure of a person, variable can be declared as:

```
struct person
{
   char name[50];
   int citNo;
   float salary;
};

int main()
{
   struct person person1, person2, person3[20];
   return 0;
}
```

Another way of creating a structure variable is:

```
struct person
{
   char name[50];
   int citNo;
   float salary;
} person1, person2, person3[20];
```

In both cases, two variables person1, person2 and an array person3 having 20 elements of type **struct person** are created.

Example-
Write a C program to add two distances entered by user. Measurement of distance should be in inch and feet. (Note: 12 inches = 1 foot)

```
#include <stdio.h>
struct Distance
{
   int feet;
   float inch;
} dist1, dist2, sum;

int main()
```

```
{
   printf("1st distance\n");

   // Input of feet for structure variable dist1
   printf("Enter feet: ");
   scanf("%d", &dist1.feet);

   // Input of inch for structure variable dist1
   printf("Enter inch: ");
   scanf("%f", &dist1.inch);

   printf("2nd distance\n");

   // Input of feet for structure variable dist2
   printf("Enter feet: ");
   scanf("%d", &dist2.feet);

   // Input of feet for structure variable dist2
   printf("Enter inch: ");
   scanf("%f", &dist2.inch);

   sum.feet = dist1.feet + dist2.feet;
   sum.inch = dist1.inch + dist2.inch;

   if (sum.inch > 12)
   {
         //If inch is greater than 12, changing it to feet.
      ++sum.feet;
      sum.inch = sum.inch - 12;
   }

   // printing sum of distance dist1 and dist2
   printf("Sum of distances = %d\'-%.1f\"", sum.feet, sum.inch);
   return 0;
}
```

Output

```
1st distance
Enter feet: 12
Enter inch: 7.9
2nd distance
Enter feet: 2
Enter inch: 9.8
Sum of distances = 15'-5.7"
```

Keyword typedef while using structure

Writing struct structure_name variable_name; to declare a structure variable isn't intuitive as to what it signifies, and takes some considerable amount of development time.
So, developers generally use typedef to name the structure as a whole. For example:

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

int main()
{
  comp comp1, comp2;
}
```

Here, typedef keyword is used in creating a type comp (which is of type as struct complex).
Then, two structure variables comp1 and comp2 are created by this comp type.

Structures within structures

Structures can be nested within other structures in C programming.

```
struct complex
{
 int imag_value;
 float real_value;
};

struct number
{
   struct complex comp;
   int real;
} num1, num2;
```

Suppose, you want to access imag_value for num2 structure variable then, following structure member is used.

```
num2.comp.imag_value
```