Structure in C:

* The structure is a user-defined data type that allows storing multiple types of data in a single unit. It occupies the sum of the memory of all members.
* The structure members can be accessed only through structure variables.
* Structure variables accessing the same structure but the memory allocated for each variable will be different.

A structure in a C program is a collection of heterogenous data clubbed together into a single entity. To define and declare a structure, C language provides the following methods:

* **Global Declaration:**

*Syntax:*

1. **struct** structure\_name
2. {
3. member definitions;
4. }structure\_variables;

*Example:*

1. **struct** Person
2. {
3. **int** age;
4. **char** name[25];
5. }p1,p2,p3;

* **Local Definiton with global declaration:**

*Syntax:*

1. **struct** structure\_name
2. {
3. member definitions;
4. };
6. **int** main()
7. {
8. **struct** structure\_name structure\_variables;
9. program\_code;
10. **return** 0;
11. }

*Example:*

1. **struct** Person
2. {
3. **int** age;
4. **char** name[25];
5. };
7. **int** main()
8. {
9. **struct** Person p1,p2,p3;
10. /\*program code\*/
11. **return** 0;
12. }

* **Using typedef**

*Syntax:*

1. **typedef** **struct** structure\_name
2. {
3. member definitions;
4. }structure\_variables;

*Example:*

1. **typedef** **struct** Person
2. {
3. **int** age;
4. **char** name[25];
5. }p1,p2,p3;

*Syntax:*

1. **typedef** **struct** structure\_name
2. {
3. member definitions;
4. };
6. **int** main()
7. {
8. structure\_name structure\_variables;
9. program\_code;
10. **return** 0;
11. }

*Example:*

1. **typedef** **struct** Person
2. {
3. **int** age;
4. **char** name[25];
5. };
7. **int** main()
8. {
9. Person p1,p2,p3;
10. /\*program code\*/
11. **return** 0;
12. }

**Note:** Notice the difference when “typedef” is used and when it is not. The keyword “struct” is not used in local declaration of structure variables with typedef.

**Accessing structure members:**

*Syntax:*

1. structure\_variable.member = value; //for structure variables
2. structure\_pointer->member = value; //for structure pointers

*Example:*

1. **typedef** **struct** Person
2. {
3. **int** age;
4. **char** name[25];
5. };
7. **int** main()
8. {
9. Person p1;
10. Person \*p2;
11. p1.age = 10;
12. p2->age = 20;
13. /\* rest of program code \*/
14. **return** 0;
15. }

There are only two ways to declare: forward declaration (which declares, but doesn’t define the struct) and definition (which declares and defines the struct):

1. **struct** y; // forward declaration
2. **struct** x { **struct** y \*p; /\* ... \*/ };
3. **struct** y { **struct** x \*q; /\* ... \*/ }; // definition

See [http://en.cppreference.com/w/c/l...](http://en.cppreference.com/w/c/language/struct) for many details

As for initialization, the syntax is the equals sign followed by a brace-enclosed list of initializers, which can be expressions, designated initializers, or nested brace-enclosed lists

1. **struct** point {**double** x,y,z;} p = {1.2, 1.3};
2. div\_t answer = {.quot = 2, .rem = -1 };
3. **struct** demo { **int** n, a[2];} z = {0, {[1] = 1}};

This can get pretty complicated when there are nested structs, unions, and arrays, see [http://en.cppreference.com/w/c/l...](http://en.cppreference.com/w/c/language/struct_initialization) for many details.

It is also possible to omit the initializer, in which case the struct will be zero-initialized if it is static and initialized to garbage values if it is local: see [implicit initialization](http://en.cppreference.com/w/c/language/initialization#Implicit_initialization)

There are two ways to declare your structure.

1st is

struct structure name

{

datatype1 variable 1;

datatype2 variable 2;

etc

}s;

Here s becomes your structure variable.

2nd ways is

struct structure name

{

datatype1 variable 1;

datatype1 variable2;

etc

};

void main()

{

struct structure name structure variable;//structure declaration

}

In the second case the declaration part is always inside the main function.

struct name\_of\_the\_Structure{

structure members

};

ex-

struct Student{

int roll;

String name;

};

struct Student s1,\*s2;

In the above example we have created a structure named as Student, with members as roll, name.If you want to use the structure members you have to create a structure variable which will initialize your members and can be accessed using dot(.) operator. Further there is a lot more application of structure in Data structure, there you will use ‘ —>’ operator to access structure members. Arrow operator is used in case of Structure pointers and is used to access the structure members when we use pointer variable to access it, ex-s2.

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows −

1. **struct** [structure tag]
2. {
3. member definition;
4. member definition;
5. ......
6. member definition;
7. }

At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables although this is purely optional and a users personal choice

1. **struct** Books
2. {
3. **char** title[50];
4. **char** author[50];
5. **char** subject[100];
6. **int** book\_id;
7. } book;

To access any member of a structure in the main function, we use the **member access operator (.)**

1. **int** main( )
2. {
3. **struct** Books Book1; /\* Declare Book1 of type Book \*/
4. **struct** Books Book2; /\* Declare Book2 of type Book \*/
5. /\* book 1 variable entry \*/
6. strcpy( Book1.title, "C Programming");
7. strcpy( Book1.author, "Nuha Ali");
8. strcpy( Book1.subject, "C Programming Tutorial");
9. }

The syntax of a C/C++ structure is as follows:

1. **struct** <structure-name>
2. {
3. //Data members
4. };

The structure name now acts as a data type. You can declare a structure variable like any other variable.

1. **<structure-name>** **<variable-name>**; //structure variable
2. **<structure-name>** \***<pointer-name>**; //structure pointer variable

You can access the data members of the structure using a period ‘.’

1. **<variable-name>**.**<data-member>**

The period is replaced with an arrow ‘->’ in case of a structure pointer variable.

Here's an example to explain the above:

1. #include<stdlib.h>
2. #include<string.h>
3. **struct** student //student is the structure's name
4. {
5. //declaring data members of student structure
6. **int** enrollmentNumber;
7. **char** name[20];
8. **float** percentage;
9. };
10. **int** main()
11. {
12. student st1; //declaring a structure variable
13. //Assigning data to st1 variable
14. st1.enrollmentNumber=30;
15. strcpy(st1.name,"Nemo");
16. st.percentage=85.35;
17. //Initializing a variable at the time of declaration
18. student st2={68,"Dory",62.87}; //Here the data 68, Dory, and 62.87 is assigned to enrollmentNumber, name, and percentage of st2 respectively
19. **return** 0;
20. }

Structures are the non-primitive, linear data structures. It is basically defined as the ***Collection of heterogenous types of data***. It is quite opposite to Arrays.

So,

How do you create a Structure ? **or**How do you define a Structure ?

Structures can be defined in two ways. They are :-

1. Using ***typedef***keyword
2. Using ***struct***keyword

***Syntax:-***

1. ***Typedef***
2. **typedef** **struct**
3. {
4. Datatype1 member\_name;
5. Datatype2 member\_name;
6. DatatypeN member\_name;
7. }structure\_name;
8. **void** main ()
9. {
10. structure\_name variable\_name;
11. }

2. ***Struct***

1. **struct** structure\_name
2. {
3. Datatype1 member\_name;
4. Datatype2 member\_name;
5. DatatypeN member\_name;
6. };
7. **void** main ()
8. {
9. **struct** structure\_name variable\_name;
10. }

Its better if you use **typedef**format because most of the times when you’re executing C progams using **struct**format then there will be chances of runtime error. The reason is because C programming language sometimes require the older versions of it to execute programs properly.