Introduction to Structure

Structure is the collection of variables of different data types (heterogeneous elements) under a single name.

Structure is a user-defined data type in C which allows you to combine different data types to store in a particular type of record. For Ex. Student detail (name, roll no, per,dob, etc) Structure helps to construct a complex data type in more meaningful way. It is somewhat similar to an Array. The only difference is that array is used to store collection of similar data types while structure can store collection of hetrogenous type of data.

Structure is used to represent a record. Suppose you want to store record of Student which consists of student name, address, roll number and age. You can define a structure to hold this information.

<u>Define and declare a structure variable:</u>

To define a structure **struct** keyword is used. struct define a new data type which is a collection of hetrogenous (different) type of data.

<u>Declaring a Structure Variables:-</u>

It is possible to declare variables of a structure, after the structure is defined. Structure variable declaration is similar to the declaration of variables of any other data types.

The syntax to define structure and to declare a variable is :

The one way is this

```
struct  tag_name
{
data type member1;
data type member2;
.
..
.
data type member N;
};

struct tag_name  V1,V2,……,Vn;
```

OR

```
struct  tag_name
{
data type member1;
data type member2;
.
..
.
data type member N;
} V1,V2,……,Vn;
```

In the above syntax two methods are given use any one to define and declare a structure.

Where tag_name is name given to a collection for ex student, book, emp etc.

Data type is any primary or fundamental data type .

Member1……N are variable names for ex. Name, Per, Age, Address, etc.
V1,V2,…..Vn are Structure variable.

Example of Structure

1) struct Book
```
{
 char name[15];
 int price;
 int pages;
   } b1,b2;
```
Here the struct Book declares a structure to hold the details of book which consists of three data members, namely name, price and pages. These members are called structure data elements or members. Each member can have different data type, like name of char type and price and pages is of int type. Book is tag name. b1 and b2 are structure variable.

2) struct Student
```
  {
   Char name[20];
    int age;
   int roll_no;
   } ;

 struct Student S1 , S2;   //declaring structure variables of Student
```

3) struct Student
```
{
 Char name[20];
 int age;
 int roll_no;
   } S1, S2 ;
```
Here S1 and S2 are variables of type structure Student.

<u>Accessing Structure Members:</u>

Structure members can be accessed and assigned values in number of ways. Structure member has no meaning independently. In order to assign a value to a structure member. The member name must be linked with the structure variable using **dot . operator** also called **period or member access operator.**

```
struct Book
{
 char name[15];
 int price;
 int pages;
} b1 , b2 ;
```

b1.price=200;   //b1 is variable of Book type and price is member of Book
We can also use scanf() to give values to structure members through terminal.
scanf(" %s ", **b1.name**);
scanf(" %d ", &**b1.price**);

Structure Initialization:-

Like any other data type, 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;
```

Example :

Program to declare a structure student with field name and roll no. take input from user and display them.

```
#include<stdio.h>
#include<conio.h>
struct student
{
 char name[10];
 int roll;
};
void main()
{
 struct student s1;
 clrscr();
 printf("\n Enter student record\n");
 printf("\n student name\t");
 scanf("%s",s1.name);
 printf("\nEnter student roll\t");
 scanf("%d",&s1.roll);
printf("\n Student detail is \n");
printf("\nstudent name is %s",s1.name);
 printf("\nroll is %d",s1.roll);
getch( );
}
```

Array of Structure:

Structure is used to store the information of One particular object but if we need to store such 100 objects then Array of Structure is used. A structure is a collection of members of different data types stored in contiguous memory locations. An array of structures is an array in which each element is a structure. This concept is very helpful in representing multiple records of a file, where each record is a collection of dissimilar data items. As we have an array of integers, we can have an array of structures also. For example, suppose we want to store the information of class of students, consisting of name, roll_number and marks,

Syntax for array of structure is

```
  struct  tag_name
  {
  data type member1;
  data type member2;
  .
  ..
  .
  data type member N;
  } array_Var[size];
```

Where tag_name is name given to a collection for ex student, book, emp etc.
Data type is any primary or fundamental data type .
Member1……N are variable names for ex. Name, Per, Age, Address, etc.
array_var are array of Structure variable.

Example :

```
struct Bookinfo
{
     char[20] bname;
     int pages;
     int price;
}b[3];
```

Explanation :
Here Book structure is used to Store the information of one Book.
In the above example if we need to store the Information of 3 books then Array of Structure is used.
b[0] stores the Information of 1st Book , b[1] stores the information of 2nd Book and So on.

Accessing Pages field of Second Book :
b[1].pages

Example :Program for storing and displaying book detail using array of structure.

```
#include <stdio.h>
struct Bookinfo
{
      char bname[20];
      int pages;
      int price;
}b[3];

void main()
{
int i;
for(i=0;i<3;i++)
    {
    printf("\nEnter the Name of Book    : ");
    gets(b[i].bname);
    printf("\nEnter the Number of Pages : ");
    scanf("%d",b[i].pages);
    printf("\nEnter the Price of Book   : ");
    scanf("%f",b[i].price);
    }

printf("\n--------- Book Details ------------ ");

for(i=0;i<3;i++)
    {
    printf("\nName of Book    : %s",b [i].bname);
    printf("\nNumber of Pages : %d",b [i].pages);
    printf("\nPrice of Book   : %f \n",b [i].price);
    }
 }
```

Output of the Structure Example:

```
Enter the Name of Book    : ABC
Enter the Number of Pages : 100
Enter the Price of Book   : 200

Enter the Name of Book    : EFG
Enter the Number of Pages : 200
Enter the Price of Book   : 300

Enter the Name of Book    : HIJ
Enter the Number of Pages : 300
Enter the Price of Book   : 500

--------- Book Details ------------

Name of Book    : ABC
Number of Pages : 100
Price of Book   : 200

Name of Book    : EFG
```

```
Number of Pages : 200
Price of Book   : 300

Name of Book    : HIJ
Number of Pages : 300
Price of Book   : 500
```

<u>Structure within Structure : Nested Structure:-</u> Structure written inside another structure is called as nesting of two structures. Nested Structures are allowed in C Programming Language. We can write one Structure inside another structure as member of another structure.

C provides us the feature of nesting one structure within another structure by using which, complex data types are created. For example, we may need to store the address of an entity employee in a structure. The attribute address may also have the subparts as street number, city, state, and pin code. Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee.

<u>Way 1</u> : Declare two separate structures

```
struct date
{
  int date;
  int month;
  int year;
};
struct Employee
  {
  char ename[20];
  int ssn;
  float salary;
  struct date doj;   //declare structure variable of date stucture
}emp1;
```

<u>Accessing Nested Elements :</u>

Structure members are accessed using dot operator.

'date' structure is nested within Employee Structure. Members of the 'date' can be accessed using 'employee'. emp1 & doj are two structure Variables names.

<u>Accessing Nested Members :</u>

Accessing Month Field : **emp1.doj.month**
Accessing day Field : **emp1.doj.day**
Accessing year Field : **emp1.doj.year**

<u>Way 2</u> : Declare Embedded structures

```
struct Employee
{
  char ename[20];
  int ssn;
  float salary;
  struct date
     {
     int date;
     int month;
     int year;
     }doj;
}emp1;
```

Accessing Nested Members :

Accessing Month Field : **emp1.doj.month**
Accessing day Field : **emp1.doj.day**
Accessing year Field : **emp1.doj.year**

Example :

```
#include <stdio.h>
struct Employee
{
   char ename[20];
   int ssn;
   float salary;
   struct date
       {
       int date;
       int month;
       int year;
       }doj;
}emp = {"Pritesh",1000,1000.50,{22,6,1990}};

Void main()
{
printf("\nEmployee Name   : %s",emp.ename);
printf("\nEmployee SSN    : %d",emp.ssn);
printf("\nEmployee Salary : %f",emp.salary);
printf("\nEmployee DOJ    : %d/%d/%d", \
        emp.doj.date, emp.doj.month, emp.doj.year);
    }
```

Output :

```
Employee Name   : Priti
Employee SSN    : 1000
Employee Salary : 10000
Employee DOJ    : 22/6/1990
```

Union:

Unions are quite similar to the structures in C. Union is also a derived type as structure. Union can be defined in same manner as structures just the keyword used in defining union in union where keyword used in defining structure was struct.

Syntax:

```
union  tag name
{
data type member1;
data type member2;
.
..
.
data type member N;
}var1,var2,.....varN;
```

Example:

```
union car{
  char name[50];
  int price;
};
```

Union variables can be created in similar manner as structure variable.

```
union car{
  char name[50];
  int price;
}c1, c2;
```

OR;

```
union car{
  char name[50];
  int price;
};
-------Inside Function-----------
union car c1, c2;
```

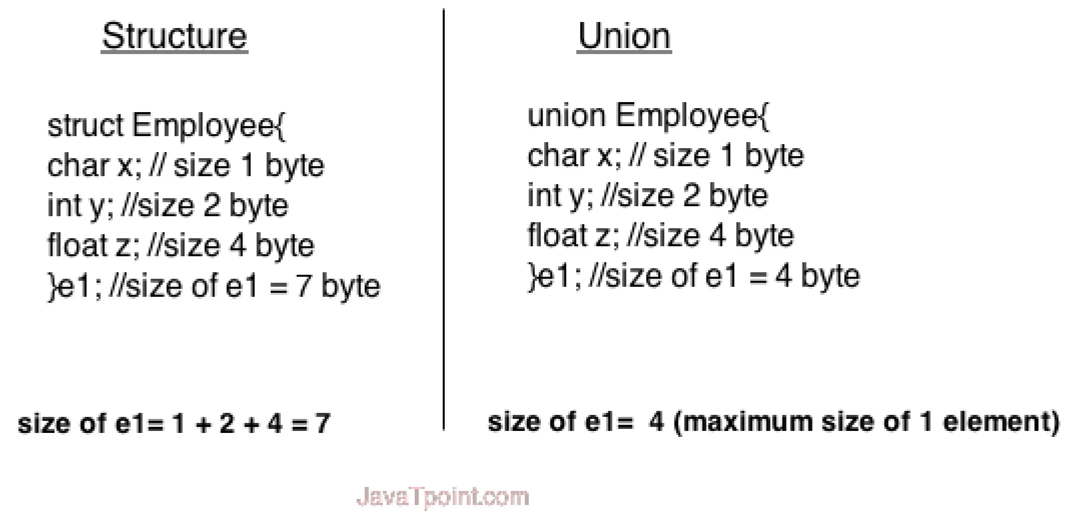
In both cases, union variables c1, c2 and union pointer variable c3 of type union car is created.

Accessing members of an union

The member of unions can be accessed in similar manner as that structure. Suppose, we you want to access price for union variable c1 in above example, it can be accessed as **c1.price**. If you want to access price for union variable c2, it can be accessed as **c2.price**.

Difference between union and structure

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.



```
struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
```

size of e1= 1 + 2 + 4 = 7

```
union Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 4 byte
```

size of e1= 4 (maximum size of 1 element)

Though unions are similar to structure in so many ways, the difference between them can be demonstrated by this example:

```
#include <stdio.h>
union job {         //defining a union
   char name[32];
   float salary;
   int worker_no;
}u;

struct job1 {                    // defining a structure
   char name[32];
   float salary;
   int worker_no;
}s;

int main()
{
   printf("size of union = %d",sizeof(u));
   printf("\nsize of structure = %d", sizeof(s));
   return 0;
}
```

Output

```
size of union = 32
size of structure = 40
```

There is difference in memory allocation between union and structure as suggested in above example. The amount of memory required to store a structure variables is the sum of memory size of all members.

But, the memory required to store a union variable is the memory required for largest element of an union.

What difference does it make between structure and union?
As you know, all members of structure can be accessed at any time. But, only one member of union can be accessed at a time in case of union and other members will contain garbage value.
Program to explain the use of union.

```
#include <stdio.h>
union job {
   char name[32];
   float salary;
   int worker_no;
} u;
void main(){
   printf("Enter name:\n");
   scanf("%s",&u.name);
   printf("Enter salary: \n");
   scanf("%f",&u.salary);
   printf("Displaying\nName :%s\n",u.name);
   printf("Salary: %.1f",u.salary);
   }
```

Output

```
Enter name
 Hillary
Enter salary
10000.00

Displaying
Name: f%Bary
Salary: 10000.00
```

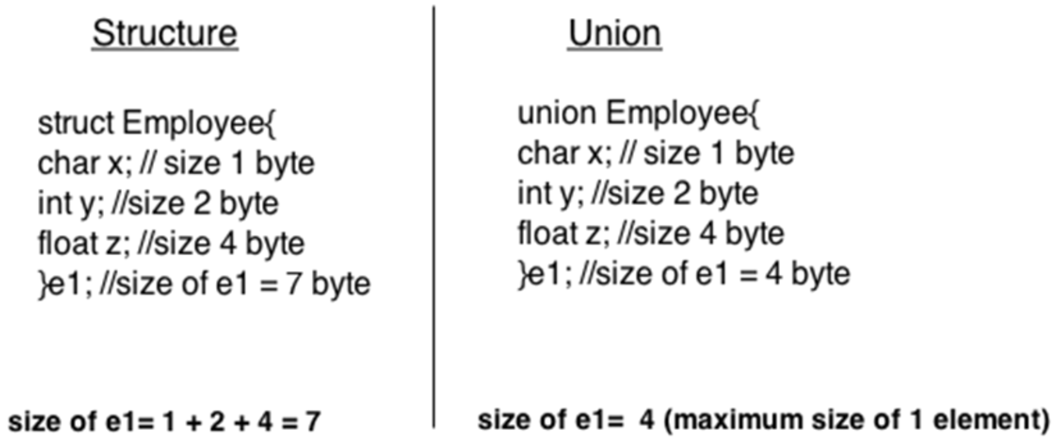
Note: You may get different garbage value of name.

Why this output?
Initially, Hillary will be stored in **u.name** and other members of union will contain garbage value. But when user enters value of salary, 10000.00 will be stored in **u.salary** and other members will contain garbage value. Thus in output, salary is printed accurately but, name displays some random string.

Difference between Structure and Union

	Structure	Union
1.	Structure is declare with the keyword 'struct'.	Union is declare with the keyword 'union'.
2.	In the structure at a time all fields are active.	In the union at a time only one field is active.
3.	In the structure all members allocate different memory location.	In the union all members share the same memory location.
4.	Total size of the structure is the sum of size of the individual member of the structure.	The size of the union is the size of the largest member in the union.

With the above example write down syntax and example of structure and union.
At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.
And Total size of the structure is the sum of size of the individual member of the structure.



Structure

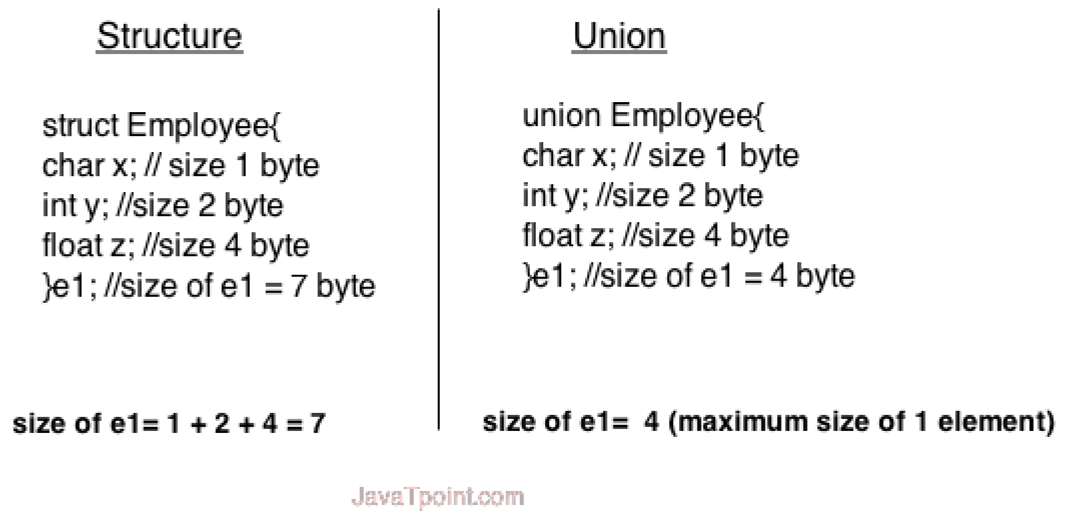
```
struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
```

size of e1= 1 + 2 + 4 = 7

Union

```
union Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 4 byte
```

size of e1= 4 (maximum size of 1 element)



Structure

```
struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
```

size of e1= 1 + 2 + 4 = 7

Union

```
union Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 4 byte
```

size of e1= 4 (maximum size of 1 element)