# What is Structure?

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member.

The **struct** keyword is used to define the structure.

Let's see the syntax to define the structure in c.

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```

Let's see the example to define a structure for an entity employee in c.

```
struct employee
{   int id;
    char name[20];
    float salary;
};
```

# Union in C

**Union** can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location.

Union is a user-defined data type, but unlike structures, they share the same memory location.

## Syntax of Union

```
union [union name]
    {
type member_1;
```

```
    type member_2;
    . . .
    type member_n;
    };
```

## Example

```
union employee
{
    string name;
    string department;
    int phone;
    string email;
};
```

## Difference between structure and union

| | STRUCTURE | UNION |
|---|---|---|
| **Keyword** | The keyword **struct** is used to define a structure | The keyword **union** is used to define a union. |
| **Size** | When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.** | when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.** |
| **Memory** | Each member within a structure is assigned unique storage area of location. | Memory allocated is shared by individual members of union. |
| **Value Altering** | Altering the value of a member will not affect other members of the structure. | Altering the value of any of the member will alter other member values. |
| **Accessing members** | Individual member can be accessed at a time. | Only one member can be accessed at a time. |
| **Initialization of Members** | Several members of a structure can initialize at once. | Only the first member of a union can be initialized. |

# C Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

```
1. int n = 10;
2. int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integ
   er.
```

# Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

```
1. int *a;//pointer to int
2. char *c;//pointer to char
```

Let's see the pointer example as explained for the above figure.

```
1. #include<stdio.h>
2. int main(){
3. int number=50;
4. int *p;
5. p=&number;//stores the address of number variable
6. printf("Address of p variable is %x \n",p); // p contains the address of the number therefore printi
   ng p gives the address of number.
7. printf("Value of p variable is %d \n",*p); // As we know that * is used to dereference a pointer th
   erefore if we print *p, we will get the value stored at the address contained by p.
8. return 0;
9. }
```

**Output**

```
Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```

## Pointer to array

```
1. int arr[10];
2. int *p[10]=&arr; // Variable p of type pointer is pointing to the address of an integer array arr.
```