UNION

- Unions are conceptually similar to structures in C
- The Syntax to declare/define a Union is also similar to that of a structure.
- The only differences is in terms of storage.
- In structure each member has its own storage location whereas all member of union uses a single shared memory location which is equal to the size of its largest data member.

Structure	Union

struct Emp
{
char x;//size 1 byte
float y;//size 4 byte
}e;
	struct Exp
	{
	char x;//size 1 byte
	float y;//size 4 byte
	}e;

- This implies that although a union may contain many members of different types, it cannot handle all the members at the same time.

Declaring a Union in C

- A Union is declared using the union keyword in C.
- Other members of Union will share the same memory address.
- To define variables of a Union, we can use Union keyword as follows,

Union item it2, it3;

Accessing a Union member in C.

- We use member access operator(.) to access members of a union in C.
- It is used between the union variable name and the union member that we want to access.
- Syntax for accessing any union member is similar to accessing structure members.

```
Union test
{
int a;
float b;
char c;
}t;
t.a;          //to access member of union t
t.b;
t.c;
```

- In unions, if we change the value of anyone member, the value of other members gets affected.

Using Union in C program

- Here is a program to understand how compiler decides size of a Union.
- The syntax is as follows

```
Union tag_name
{
member definition;
member definition;
……..
member definition;
}
Union variable(s);
```

Example:

```
Union item
{
int m;
float x;
char c;
}It1;
```

- This declares a variable IT1, of type Union item.
- This union contains three members each with a different datatype.
- However only one of them can be used at a time.
- This is due to the fact only one location is allocated for all the union variable, irrespective of their size.
- The compiler allocates the storage that is large enough to hold the largest variable type in the union.
- In the union declared above the member X requires 4 bytes which is largest amongst the members for a 16 bit machine

Example

```
#include<stdio.h>
#include<conio.h>
Union one
{
int x;
char y;
}one 1;
Union two
{
int x;
char y;
longz;
}two 2;
Union three
```

```
{
int arr[100];
char y;
double d[5];
}three 3;
}
int main()
{
printf("size of (one)=%lu, size of (two)=%lu, size of (three)=%lu,
        size of (one1)=%lu, size of (two 2)=%lu, size of (three 3)=%lu,");
return 0;
}
```

Output

Size of (one)=4, Size of (two)=8, Size of(three)=400

Let's see another code example,

```
//defining and printing members of a Union
#include<stdio.h>
Union item
{
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter record %d %f %c", &it[i].a,&it[i].b,&it[i].ch);
}
for(i=0;i<n;i++)
{
printf("\nrecord no.%d:\n",i+1);
printf("%d%f%c",it[i].a,it[i].b,it[i].ch);
}return0;
}
```

Output

```
Enter the number of records : 2
Enter record 1:
1
3
A
Enter record 2:
2
4
```

D
Record No 1:
1082130532
4.000048
D

As you can see here, the values of int and float get corrupted and only char variable prints the expected result. This is because in Union, the memory is shared among different data types.

In the above example, value of the char variable was stored at last, hence the value of other variables is last.

Difference Between Structure ad Union in C

Structure	**Union**
In a structure we can initialize many data members at once.	In union, we can only initialize the first data member
Compiler allocates memory for each member of a structure.	While for a union, t allocates memory equal to the size f the largest data member.
Structure members have a unique storage location each	Union members share a memory location
In a structure we can access individual members simultaneously	In union, we can only access one member at a time.
If we change the value of a member in a structure, it won't affect its other members.	In a Union, changing the value of one member will affect the others.