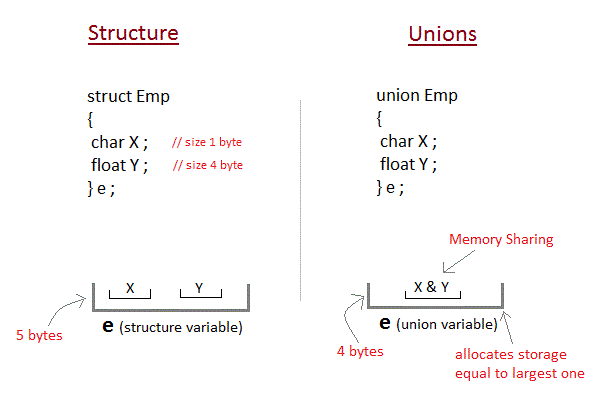
**Unions in C Language**

**Unions** are conceptually similar to **structures**. 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 members of **union** uses a single shared memory location which is equal to the size of its largest data member.

**Note :** we can use all members of structure at once. But in case of union we can only use any one member at once because of shared memory location.

If you try to use all members of union together , you will get unexpected valued back from members except last member.



Ex :

#include <stdio.h>

union item

{

int a;

float b;

char ch;

};

int main( )

{

union item it;

it.a = 12;

it.b = 20.2;

it.ch = 'z';

printf("%d\n", it.a);

printf("%f\n", it.b);

printf("%c\n", it.ch);

return 0;

}

**Expected output :**

12

20.2

z

**Actual output:**

1101109626 //corrupted

20.199940 //corrupted

Z

As you can see here, the values of a and b get corrupted and only variable c prints the expected result. This is because in union, the memory is shared among different data types. Hence, the only member whose value is currently stored will have the memory.

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

**When to use unions**

Unions can be useful when we want to use same memory for two or more members.

EX : Struct with three different variables share same value. No need to initialize three variables, initialize one variable and use it for other variables too.

**Sizeof struct and union**

#include <stdio.h>

struct employee{

int age;

int salary;

};

union student{

int age;

int scholarship;

};

int main()

{

struct employee e;

union student s;

printf("%d\n",sizeof(e)); //8bytes

printf("%d\n",sizeof(s)); //4bytes - members share same memory location

return 0;

}