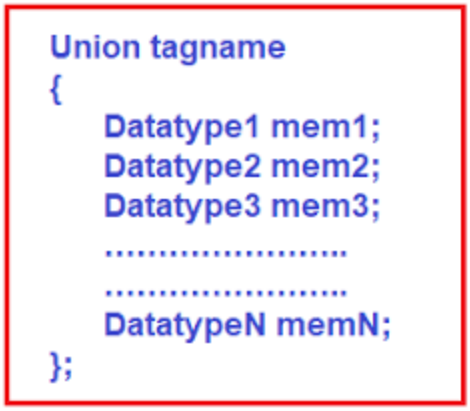
C Unions

The Union is a user-defined data type in C language that can contain elements of the different data types just like structure. But unlike structures, all the members in the C union are stored in the same memory location. Due to this, only one member can store data at the given instance.



Syntax of Union in C

The syntax of the union in C can be divided into three steps which are as follows:

C Union Declaration

In this part, we only declare the template of the union, i.e., we only declare the members' names and data types along with the name of the union. No memory is allocated to the union in the declaration.

```
union union_name {
   datatype member1;
   datatype member2;
   ...
};
```

Keep in mind that we have to always end the union declaration with a semi-colon. Also, If you're looking to explore unions and their use in data structures, the **C Programming Course Online with Data Structures** provides detailed explanations and practical examples.

Different Ways to Define a Union Variable

We need to define a variable of the union type to start using union members. There are two methods using which we can define a union variable.

1. With Union Declaration
2. After Union Declaration

1. Defining Union Variable with Declaration

```
union union_name {
   datatype member1;
   datatype member2;
   ...
} var1, var2, ...;
```

2. Defining Union Variable after Declaration

union *union_name var1, var2, var3*...;

where *union_name* is the name of an already declared union.

Access Union Members

We can access the members of a union by using the (.) dot operator just like structures.

var1.member1;

where *var1* is the **union variable** and *member1* is the **member of the union**.

The above method of accessing the members of the union also works for the nested unions.

var1.member1.memberA;

Here,

- *var1* is a union variable.
- *member1* is a member of the union.
- *memberA* is a member of member1.

Initialization of Union in C

The initialization of a union is the initialization of its members by simply assigning the value to it.

var1.member1 = some_value;

One important thing to note here is that **only one member can contain some value at a given instance of time.**

Example of Union

```
// C Program to demonstrate how to use union

#include <stdio.h>

// union template or declaration

union un {

   int member1;

   char member2;

   float member3;

};

// driver code
```

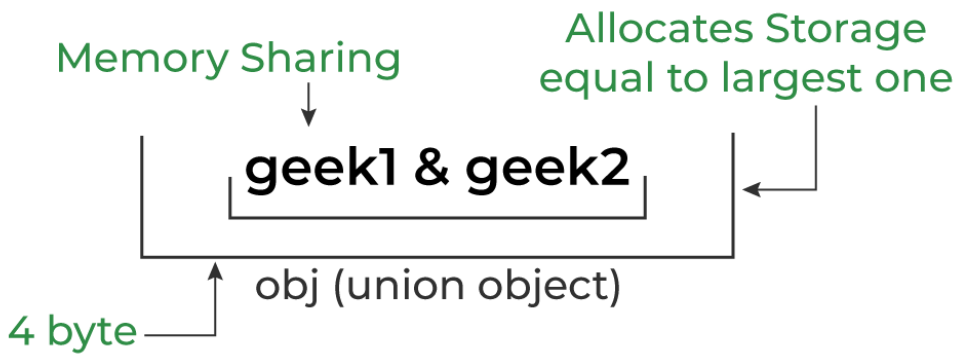
```
int main()
{
// defining a union variable
   union un var1;
// initializing the union member
   var1.member1 = 15;
printf("The value stored in member1 = %d",
         var1.member1);
return 0;
}
```

Output

The value stored in member1 = 15

Size of Union

The size of the union will always be equal to the size of the largest member of the array. All the less-sized elements can store the data in the same space without any overflow.



Example 1: C program to find the size of the union

```
// C Program to find the size of the union
#include <stdio.h>
```

```
// declaring multiple unions
union test1 {
   int x;
   int y;
} Test1;
union test2 {
   int x;
   char y;
} Test2;
union test3 {
   int arr[10];
   char y;
} Test3;
// driver code
int main()
{
   // finding size using sizeof() operator
   int size1 = sizeof(Test1);
   int size2 = sizeof(Test2);
   int size3 = sizeof(Test3);
   printf("Sizeof test1: %d\n", size1);
   printf("Sizeof test2: %d\n", size2);
   printf("Sizeof test3: %d", size3);
   return 0;
}
```

Output

```
Sizeof test1: 4
Sizeof test2: 4
Sizeof test3: 40
```

Difference between C Structure and C Union

Difference between Structure and Union

STRUCTURE	UNION
Every memory has its own memory sapce.	All the members use the same memory space to store the values.
It can handle all the members(or) a few as required at a time.	It can handle only one member at a time,as all the members use the same space.
Keyword struct is used.	Keyword union is used.
It may be initialized with all its members.	Only its first member may be initialized.
Any member can be accessed at any time without the loss of data.	Only one member can be accessed at any time with the loss of previous data.
Different interpretation for the same memory location are not possible.	Different interpretations for the same
More storage space is required.	Minimum storage space is required.
Syntax: structure strut-name { }var1..varn;	Syntax: union union-name { }var1..varn;