C - Unions

Previous Page Next Page

A **union** is a special data type available in C that enables you to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multi-purpose.

Defining a Union

To define a union, you must use the **union** statement in very similar was as you did while defining structure. The union statement defines a new data type, with more than one member for your program. The format of the union statement is as follows:

```
union [union tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

The **union tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the union's definition, before the final semicolon, you can specify one or more union variables but it is optional. Here is the way you would define a union type named Data which has the three members i, f, and str:

```
union Data
{
   int i;
   float f;
   char  str[20];
} data;
```

Now, a variable of **Data** type can store an integer, a floating-point number, or a string of characters. This means that a single variable ie. same memory location can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in above example Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by character string. Following is the example which will display total memory size occupied by the above union:

Try it

```
#include <stdio.h>
#include <string.h>

union Data
{
   int i;
   float f;
   char  str[20];
};

int main( )
{
   union Data data;

   printf( "Memory size occupied by data : %d\n", sizeof(data));

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Memory size occupied by data : 20
```

Accessing Union Members

To access any member of a union, we use the **member access operator (.)**. The member access operator is coded as a period between the union variable name and the union member that we wish to access. You would use **union** keyword to define variables of union type. Following is the example to explain usage of union:

```
#include <stdio.h>
#include <string.h>

union Data
{
   int i;
   float f;
   char  str[20];
};

int main( )
{
   union Data data;

   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "C Programming");

   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming
```

Here, we can see that values of **i** and **f** members of union got corrupted because final value assigned to the variable has occupied the memory location and this is the reason that the value if **str** member is getting printed very well. Now let's look into the same example once again where we will use one variable at a time which is the main purpose of having union:

```
#include <stdio.h>
#include <string.h>

union Data
{
   int i;
   float f;
   char  str[20];
};

int main( )
{
   union Data data;

   data.i = 10;
   printf( "data.i : %d\n", data.i);

   data.f = 220.5;
   printf( "data.f : %f\n", data.f);

   strcpy( data.str, "C Programming");
   printf( "data.str : %s\n", data.str);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

Here, all the members are getting printed very well because one member is being used at a time.