UNIONS IN C

A **union** is a special data type available in C that allows 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 multiple-purpose.

Defining a Union

To define a union, you must use the **union** statement in the same way as you did while defining a 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 having 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. It means a single variable, i.e., 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 the above example, Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string. The following example displays the total memory size occupied by the above union −

```
#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 the keyword **union** to define variables of union type. The following example shows how to use unions in a program −

```
#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 the values of **i** and **f** members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of **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 unions −

```
#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.

Union of structures

- A structure can be nested inside a union and it is called union of structures.
- It is possible to create a union inside a structure.

Sample Program 2

An another C program which shows the usage of union of structure is given below −

```
struct x{
   int a;
   float b;
};
union z{
```

```
  struct x s;
};
main ( ){
  union z u;
  u.s.a = 10;
  u.s.b = 30.5;
  printf("a=%d", u.s.a);
  printf("b=%f", u.s.b);
}
```

Output

When the above program is executed, it produces the following result –

a= 10

b = 30.5

Enum in C: Understanding The Concept of Enumeration

Enumeration or Enum in C is a special kind of data type defined by the user. It consists of constant integrals or integers that are given names by a user. The use of enum in C to name the integer values makes the entire program easy to learn,

Syntax to Define Enum in C

An enum is defined by using the 'enum' keyword in C, and the use of a comma separates the constants within. The basic syntax of defining an enum is:

enum enum_name{int_const1, int_const2, int_const3, …. int_constN};

In the above syntax, the default value of int_const1 is 0, int_const2 is 1, int_const3 is 2, and so on. However, you can also change these default values while declaring the enum. Below is an example of an enum named cars and how you can change the default values.

enum cars{BMW, Ferrari, Jeep, Mercedes-Benz};

Here, the default values for the constants are:

BMW=0, Ferrari=1, Jeep=2, and Mercedes-Benz=3. However, to change the default values, you can define the enum as follows:

```
enum cars
 {
BMW=3,
Ferrari=5,
Jeep=0,
Mercedes-Benz=1
};
```

Enumerated Type Declaration to Create a Variable

Similar to pre-defined data types like int and char, you can also declare a variable for enum and other user-defined data types. Here's how to create a variable for enum.

enum condition (true, false); //declaring the enum

enum condition e; //creating a variable of type condition

Suppose we have declared an enum type named condition; we can create a variable for that data type as mentioned above. We can also converge both the statements and write them as:

enum condition (true, false) e;

For the above statement, the default value for true will be 1, and that for false will be 0.

How to Create and Implement Enum in C Program

Now that we know how to define an enum and create variables for it, let's look at some examples to understand how to implement enum in C programs.

Example 1: Printing the Values of Weekdays

```
#include <stdio.h>
   enum days {Sunday=1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
int main(){
   // printing the values of weekdays
   for(int i=Sunday;i<=Saturday;i++)
{
      printf("%d, ",i);
   }
   return 0;
}
```

Output:

```
1, 2, 3, 4, 5, 6, 7,
```

In the above code, we declared an enum named days consisting of the name of the weekdays starting from Sunday. We then initialized the value of Sunday to be 1. This will assign the value for the other days as the previous value plus 1. To iterate through the enum and print the values of each day, we have created a for loop and initialized the value for i as Sunday.