Enum in C

The enum in C is also known as the enumerated type. It is a user-defined data type that consists of integer values, and it provides meaningful names to these values. The use of enum in C makes the program easy to understand and maintain. The enum is defined by using the enum keyword.

The following is the way to define the enum in C:

enum flag{integer\_const1, integer\_const2,.....integter\_constN};

n the above declaration, we define the enum named as flag containing 'N' integer constants. The default value of integer\_const1 is 0, integer\_const2 is 1, and so on. We can also change the default value of the integer constants at the time of the declaration.

enum fruits{mango, apple, strawberry, papaya};

The default value of mango is 0, apple is 1, strawberry is 2, and papaya is 3. If we want to change these default values, then we can do as given below:

enum fruits{

mango=2,

apple=1,

strawberry=5,

papaya=7,

};

Enumerated type declaration

As we know that in C language, we need to declare the variable of a pre-defined type such as int, float, char, etc. Similarly, we can declare the variable of a user-defined data type, such as enum. Let's see how we can declare the variable of an enum type.

Suppose we create the enum of type status as shown below:

enum status{false,true};

Now, we create the variable of status type:

enum status s; // creating a variable of the status type.

In the above statement, we have declared the 's' variable of type status.

To create a variable, the above two statements can be written as:

enum status{false,true} s;

In this case, the default value of false will be equal to 0, and the value of true will be equal to 1.

1. #include <stdio.h>
2. **enum** weekdays{Sunday=1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
3. **int** main()
4. {
5. **enum** weekdays w; // variable declaration of weekdays type
6. w=Monday; // assigning value of Monday to w.
7. printf("The value of w is %d",w);
8. **return** 0;
9. }

In the above code, we create an enum type named as weekdays, and it contains the name of all the seven days. We have assigned 1 value to the Sunday, and all other names will be given a value as the previous value plus one.

**Ex:**

1. include <stdio.h>
2. **enum** months{jan=1, feb, march, april, may, june, july, august, september, october, november, december};
3. **int** main()
4. {
5. // printing the values of months
6. **for**(**int** i=jan;i<=december;i++)
7. {
8. printf("%d, ",i);
9. }
10. **return** 0;
11. }

n the above code, we have created a type of enum named as months which consists of all the names of months. We have assigned a '1' value, and all the other months will be given a value as the previous one plus one. Inside the main() method, we have defined a for loop in which we initialize the 'i' variable by jan, and this loop will iterate till December.

Ex:

1. #include <stdio.h>
2. **enum** days{sunday=1, monday, tuesday, wednesday, thursday, friday, saturday};
3. **int** main()
4. {
5. **enum** days d;
6. d=monday;
7. **switch**(d)
8. {
9. **case** sunday:
10. printf("Today is sunday");
11. **break**;
12. **case** monday:
13. printf("Today is monday");
14. **break**;
15. **case** tuesday:
16. printf("Today is tuesday");
17. **break**;
18. **case** wednesday:
19. printf("Today is wednesday");
20. **break**;
21. **case** thursday:
22. printf("Today is thursday");
23. **break**;
24. **case** friday:
25. printf("Today is friday");
26. **break**;
27. **case** saturday:
28. printf("Today is saturday");
29. **break**;
30. }
32. **return** 0;
33. }

f we do not provide any value to the enum names, then the compiler will automatically assign the default values to the enum names starting from 0.

We can also provide the values to the enum name in any order, and the unassigned names will get the default value as the previous one plus one.

The values assigned to the enum names must be integral constant, i.e., it should not be of other types such string, float, etc.

All the enum names must be unique in their scope, i.e., if we define two enum having same scope, then these two enums should have different enum names otherwise compiler will throw an error.

1. #include <stdio.h>
2. **enum** status{success, fail};
3. **enum** boolen{fail,pass};
4. **int** main(**void**) {
6. printf("The value of success is %d", success);
7. **return** 0;
8. }

* In enumeration, we can define an enumerated data type without the name also.

1. #include <stdio.h>
2. **enum** {success, fail} status;
3. **int** main(**void**) {
4. status=success;
5. printf("The value of status is %d", status);
6. **return** 0;
7. }

Enum vs. Macro in C

* Macro can also be used to define the name constants, but in case of an enum, all the name constants can be grouped together in a single statement.  
  For example,

# define pass 0;  
# define success 1;

The above two statements can be written in a single statement by using the enum type.  
enum status{pass, success};

* The enum type follows the scope rules while macro does not follow the scope rules.
* In Enum, if we do not assign the values to the enum names, then the compiler will automatically assign the default value to the enum names. But, in the case of macro, the values need to be explicitly assigned.
* The type of enum in C is an integer, but the type of macro can be of any type.