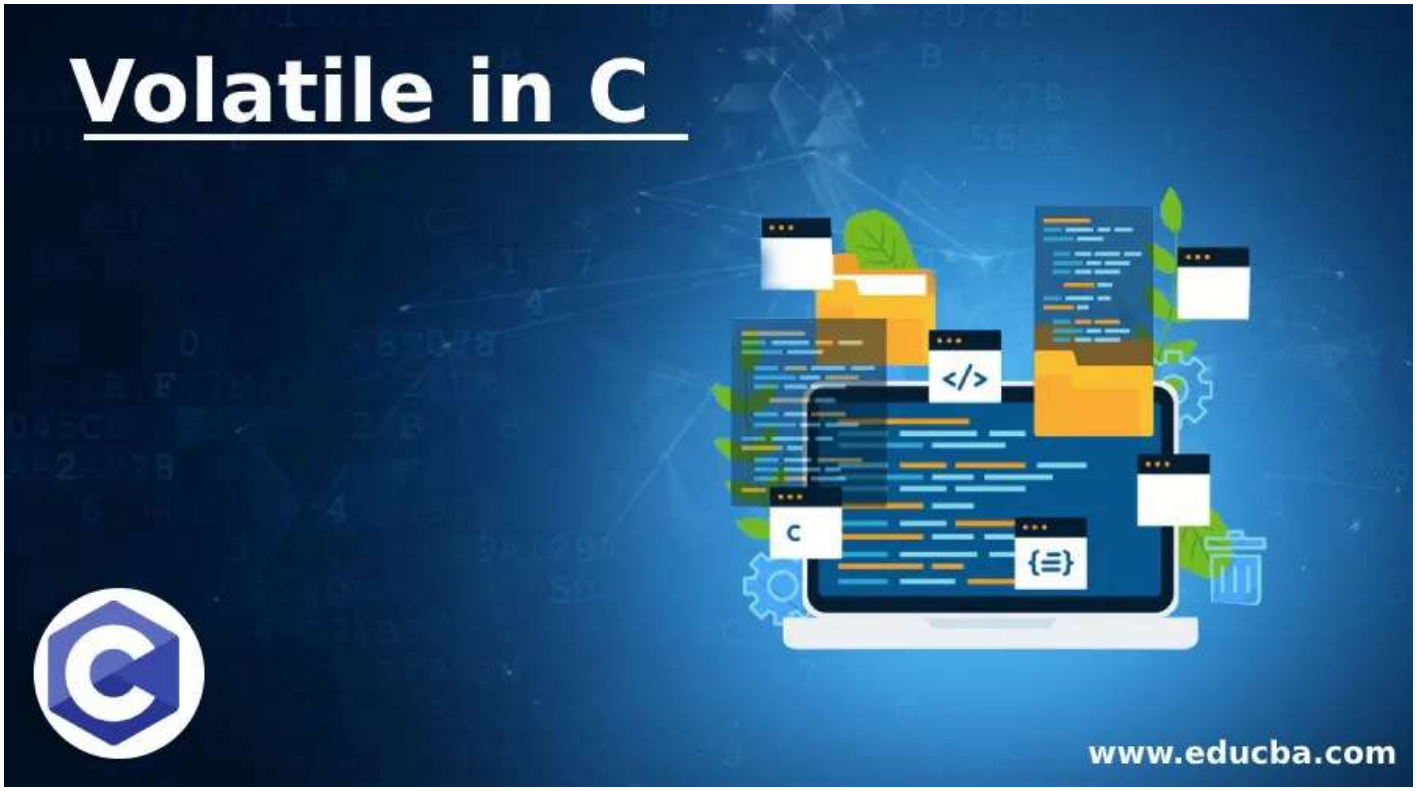

Introduction to Volatile in C

A volatile keyword in C is nothing but a qualifier that is used by the programmer when they declare a variable in source code. It is used to inform the compiler that the variable value can be changed any time without any task given by the source code. Volatile is usually applied to a variable when we are declaring it. The main reason behind using volatile keyword is that it is used to prevent optimizations on objects in our source code. Therefore, an object declared as volatile can't be optimized because its value can be easily changed by the code. As we have seen what is Volatile in C. Similarly, we will see the syntax used to represent a volatile object in code. But keep in mind that the value of volatile keyword can't be changed by the program explicitly.

Syntax

```
volatile data_type variable_name ;
volatile data_type  *variable_name ;
```

Explanation: In the above declaration volatile keyword is mandatory to be used then data_type means any data type it can be wither integer, float, or double. Finally, the name of the variable as per our choice. As both the declarations are correct we can use any of the above to declare a volatile variable.

For Example :

```
volatile int x ;
volatile int *a;
```

How Does Volatile Keyword work in C?

Now let's see how does a volatile keyword works in C programming code through some coding examples with a brief explanation. In the below two codes we will see how does the program changes when we use volatile keyword in declaring a variable as compared to the non-volatile keyword. We will see how the efficiency of the code changes when we use volatile and how quickly we can apply this functionality in our code. Volatile is used in C programming when we need to go and read the value stored by the pointer at the address pointed by the pointer. If you need to change anything in your code that is out of compiler reach you can use this volatile keyword before the variable for which you want to change the value.

Examples to Implement Volatile in C

Here is the sample code to demonstrate the working of volatile keyword:

Example #1

Without using keyword Volatile:

Code:

```
#include<stdio.h> // C header file for standard input and output
int a = 0 ; // initilaizing and declaring the integer a to value 0.
int main ()  // main class
{
```

```
if ( a == 0 )  //  This condition will be true
{
printf ( " a = 0  \n " ) ;
}
else                        // Else part will be optimized
{
printf ( " a ! = 0  \n " ) ;
}
return 0 ; // returning value
}
```

Output:

```
a = 0
```

Explanation: In the above code, we have declared an integer variable with value 0 assigned to it. Then in the main class, we have set the if condition which will hold true until and unless the value of variable a is 0. As you can see the output will always be 0 as the condition will always remain true so that that code won't move to the else part as it will ignore the else part. But things will change when we will add keyword volatile to the declaration of integer variable a. Let's have a look at the other code.

Example #2

With using keyword Volatile:

Code:

```
#include<stdio.h>
volatile int a ;    /* volatile Keyword used before declaration of
integer variable a */
int main() // main class
{
a = 0 ;   // initializing the integer value to 0
if (a == 0)  // applying if condition
{
printf ( " a = 0 \n " ) ;
}
else// Now compiler never optimize else part because the variable is
declared as volatile
{
printf ( " a ! = 0  \n " ) ;
}
return 0 ;
}
```

Output:

```
a = 0
```

Explanation: In the above code, we have declared a volatile integer variable a. Then in the main class, we have set two things one is the value of integer variable is 0 and second is the if condition which will hold true until and unless the value of variable a is 0. As you can see the output will always be 0 as the condition will always remain true because the variable is declared as volatile. Therefore, the compiler won't optimize the else part of code because of the volatile keyword used before integer. So the compiler will know that the variable can change anytime.hence, it will read the else part as the final executable code and display the result.

Example #3

Here is another C programming code to demonstrate the working of volatile keyword in C:

Code:

```
#include <stdio.h>
int main (void)                 // main class declaration in the code
{
const volatile int local_value = 25 ; // declaring constant volatile
integer variable with assigned value
int *ptr = ( int* ) &local_value ;
printf ( " The initial value of the local_value is  : %d \n ",
local_value ) ;
*ptr = 195 ;  // value to the pointer
printf ( " The modified value of the local_value is: %d \n ",
local_value ) ;
return 0 ;
}
```

Output:

```
The initial value of the local_value is  : 25
 The modified value of the local_value is: 195
```

Explanation: In the above code, you can see we have declared a constant volatile variable of an integer data type with name local_value and we allocated the value 25 to it. Then we have declared the pointer of integer data type in which we are storing the address value of “local_value”. In addition, we are printing the old value then we are printing the modified value on the screen. This modification is possible only because of the volatile keyword we have used in the declaration of the variable.

Conclusion

volatile plays an important role in C programming as the compiler can’t guess about the value. The main reason behind using volatile is that it can change value any time a user wants it to be changed or when another thread is running but using the same variable.