



## Learn, Share, Build

Each month, over 50 million developers come to Stack Overflow to learn, share their knowledge, and build their careers.

Google

Facebook

OR

Join the world's largest developer community.


## Difference between const & const volatile

If we declare a variable as `volatile` every time the fresh value is updated  
If we declare a variable as `const` then the value of that variable will not be changed

Then `const volatile int temp;`  
What is the use of declaring the variable `temp` as above?  
What happens if we declare as `const int temp` ?

c embedded

edited Apr 1 '14 at 6:05  
user2045557

asked Jan 4 '11 at 10:48  
 user559208  
425 2 8 8

You wouldn't use `const volatile int temp;` at block scope (i.e. inside `{ }`), it has no use there. – M.M  
Feb 15 '16 at 21:57

### 9 Answers

An object marked as `const volatile` will not be permitted to be changed by the code (an error will be raised due to the `const` qualifier) - at least through that particular name/pointer.

The `volatile` part of the qualifier means that the compiler cannot optimize or reorder access to the object.

In an embedded system, this is typically used to access hardware registers that can be read and are updated by the hardware, but make no sense to write to (or might be an error to write to).

An example might be the status register for a serial port. Various bits will indicate if a character is waiting to be read or if the transmit register is ready to accept a new character (ie., - it's empty). Each read of this status register could result in a different value depending on what else has occurred in the serial port hardware.

It makes no sense to write to the status register (depending on the particular hardware spec), but you need to make sure that each read of the register results in an actual read of the hardware - using a cached value from a previous read won't tell you about changes in the hardware state.

A quick example:

```
unsigned int const volatile *status_reg; // assume these are assigned to point to the
unsigned char const volatile *recv_reg; // correct hardware addresses
```

```
#define UART_CHAR_READY 0x00000001
```

```
int get_next_char()
{
    while ((*status_reg & UART_CHAR_READY) == 0) {
        // do nothing but spin
    }
}
```

```

    }

    return *recv_reg;
}

```

If these pointers were not marked as being `volatile`, a couple problems might occur:

- the while loop test might read the status register only once, since the compiler could assume that whatever it pointed to would never change (there's nothing in the while loop test or loop itself that could change it). If you entered the function when there was no character waiting in UART hardware, you might end up in an infinite loop that never stopped even when a character was received.
- the read of the receive register could be moved by the compiler to before the while loop - again because there's nothing in the function that indicates that `*recv_reg` is changed by the loop, there's no reason it can't be read before entering the loop.

The `volatile` qualifiers ensures that these optimizations are not performed by the compiler.

edited Jan 4 '11 at 18:56

answered Jan 4 '11 at 18:44



Michael Burr

262k 34 397 629

+1 for explanation. And I have a question: what about const volatile methods? If I have a class, which is accessed by many threads (although access is synchronized with mutex) does my const methods also have to be volatile (since some variable could be changed by other thread) – Sasa Aug 8 '12 at 23:31

- `volatile` will tell the compiler not to optimise code related the variable, usually when we know it can be changed from "outside", e.g. by another thread.
- `const` will tell the compiler that it is forbidden for the program to modify the variable's value.
- `const volatile` is a very special thing you'll probably see used exactly 0 times in your life (tm). As is to be expected, it means that the program cannot modify the variable's value, but the value can be modified from the outside, thus no optimisations will be performed on the variable.

answered Jan 4 '11 at 11:02



mingos

16.9k 4 52 96

7 I'd have thought that `volatile` variables are usually what happens when you start messing with hardware, not with other threads. Where I've seen `const volatile` used is in things like memory-mapped status registers or the like. – JUST MY correct OPINION Jan 4 '11 at 12:33

1 Of course, you're absolutely right, multithreading is just one example, but not the only one :). – mingos Jan 4 '11 at 15:16

12 If you work with embedded systems you will see this very often. – Daniel Grillo Oct 11 '13 at 12:04

It is not because the variable is `const` that it may not have changed between two sequence points.

Constness is a promise you make not to change the value, not that the value won't be changed.

answered Jan 4 '11 at 10:52



Alexandre C.

40.8k 6 93 172

4 Plus one for pointing out that `const` data is not "constant". – Bogdan Alexandru Jan 26 '15 at 7:47

I've needed to use this in an embedded application where some configuration variables are located in an area of flash memory that can be updated by a bootloader. These config variables are 'constant' during runtime, but without the volatile qualifier the compiler would optimise something like this...

```
cantx.id = 0x10<<24 | CANID<<12 | 0;
```

...by precomputing the constant value and using an immediate assembly instruction, or loading the constant from a nearby location, so that any updates to the original CANID value in the config flash area would be ignored. CANID has to be const volatile.

answered Jun 28 '15 at 10:55

push2eject  
118 1 5

`const` means that the variable cannot be modified by the c code, not that it cannot change. It means that no instruction can write to the variable, but its value might still change.

`volatile` means that the variable may change at any time and thus no cached values might be used; each access to the variable has to be executed to its memory address.

Since the question is tagged "embedded" and supposing `temp` is a user declared variable, not a hardware-related register (since these are usually handled in a separate .h file), consider:

An embedded processor which has both volatile read-write data memory (RAM) and non-volatile read-only data memory, for example FLASH memory in von-Neumann architecture, where data and program space share a common data and address bus.

If you declare `const temp` to have a value (at least if different from 0), the compiler will assign the variable to an address in FLASH space, because even if it were assigned to a RAM address, it still needs FLASH memory to store the initial value of the variable, making the RAM address a waste of space since all operations are read-only.

In consequence:

`int temp;` is a variable stored in RAM, initialized to 0 at startup (cstart), cached values may be used.

`const int temp;` is a variable stored in (read-only)FLASH, initialized to 0 at compiler time, cached values may be used.

`volatile int temp;` is a variable stored in RAM, initialized to 0 at startup (cstart), cached values will NOT be used.

`const volatile int temp;` is a variable stored in (read-only)FLASH, initialized to 0 at compiler time, cached values will NOT be used

Here comes the usefull part:

Nowadays most Embedded processors have the ability to make changes to their read-only non-volatile memory by means of a special function module, in which case `const int temp` can be changed at runtime, although not directly. Said in another way, a function may modify the value at the address where `temp` is stored.

A practical example would be to use `temp` for the device serial number. The first time the embedded processor runs, `temp` will be equal to 0 (or the declared value) and a function can use this fact to run a test during production and if successfull, ask to be assigned a serial number and modify the value of `temp` by means of a special function. Some processors have a special address range with OTP (one-time programmable) memory just for that.

But here comes the difference:

If `const int temp` is a modifiable ID instead of a one-time-programmable serial number and is NOT declared `volatile`, a cached value might be used untill the next boot, meaning the new ID might not be valid untill the next reboot, or even worse, some functions might use the new value while other might use an older cached value untill reboot. If `const int temp` IS declared `volatile`, the ID change will take effect immediately.

edited Sep 30 '15 at 17:48

answered Sep 30 '15 at 16:38

Michael Kusch  
101 3

This article discusses the scenarios where you want to combine const and volatile qualifiers.

<http://embeddedgurus.com/barr-code/2012/01/combining-cs-volatile-and-const-keywords/>

answered Nov 15 '15 at 7:55

balajimc55



494 4 12

In C, const and volatile are type qualifiers and these two are independent.

Basically, const means that the value isn't modifiable by the program.

And volatile means that the value is subject to sudden change (possibly from outside the program).

In fact, C standard mentions an example of valid declaration which is both const and volatile. The example is

```
"extern const volatile int real_time_clock;"
```

where real\_time\_clock may be modifiable by hardware, but cannot be assigned to, incremented, or decremented.

So we should already treat const and volatile separately. Besides, these type qualifier applies for struct, union, enum and typedef as well.

answered Aug 9 '16 at 11:19

[user2903536](#)

503 7 13

You can use const and volatile together. For example, if 0x30 is assumed to be the value of a port that is changed by external conditions only, the following declaration would prevent any possibility of accidental side effects:

```
const volatile char *port = (const volatile char *)0x30;
```

answered Dec 23 '16 at 17:08

[Step](#)

87 9

We use 'const' keyword for a variable when we don't want the program to change it. Whereas when we declare a variable 'const volatile' we are telling the program not to change it and the compiler that this variable can be changed unexpectedly from input coming from the outside world.

[edited Oct 17 '16 at 4:43](#)

answered Nov 17 '15 at 3:30

[Ali](#)

29 4

This answer does not take into account the `volatile` specifier at all. – [The amateur programmer](#) Aug 20 '16 at 7:36