Why is volatile needed in C?

Why is `volatile` needed in C? What is it used for? What will it do?

c declaration volatile

edited May 27 '15 at 17:57
Jonathan Leffler
520k ●81 ●612 ●953

asked Oct 29 '08 at 8:36
thomas

1 http://stackoverflow.com/questions/72552/c-when-has-the-volatile-keyword-ever-helped-you – Frederik Slijkerman Oct 29 '08 at 8:45

15 Answers

Volatile tells the compiler not to optimize anything that has to do with the volatile variable.

There is only one reason to use it: When you interface with hardware.

Let's say you have a little piece of hardware that is mapped into RAM somewhere and that has two addresses: a command port and a data port:

```
typedef struct
{
  int command;
  int data;
  int isbusy;
} MyHardwareGadget;
```

Now you want to send some command:

```
void SendCommand (MyHardwareGadget * gadget, int command, int data)
{
  // wait while the gadget is busy:
  while (gadget->isbusy)
  {
    // do nothing here.
  }
  // set data first:
  gadget->data    = data;
  // writing the command starts the action:
  gadget->command = command;
}
```

Looks easy, but it can fail because the compiler is free to change the order in which data and commands are written. This would cause our little gadget to issue commands with the previous data-value. Also take a look at the wait while busy loop. That one will be optimized out. The compiler will try to be clever, read the value of isbusy just once and then go into an infinite loop. That's not what you want.

The way to get around this is to declare the pointer gadget as volatile. This way the compiler is forced to do what you wrote. It can't remove the memory assignments, it can't cache variables in registers and it can't change the order of assignments either:

This is the correct version:

```
   void SendCommand (volatile MyHardwareGadget * gadget, int command, int data)
    {
      // wait while the gadget is busy:
      while (gadget->isbusy)
      {
        // do nothing here.
      }
      // set data first:
      gadget->data    = data;
      // writing the command starts the action:
      gadget->command = command;
    }
```

edited Dec 12 '13 at 9:23 answered Oct 29 '08 at 8:45

Vikas Goel
335 ● 1 ● 3 ● 24

Nils Pipenbrinck
61.3k ● 19 ● 130 ● 204

31 Personally, I'd prefer the integer size to be explicity e.g. int8/int16/int32 when talking to hardware. Nice answer though ;) – tonylo Oct 29 '08 at 8:53

10 yes, you should declare things with a fixed register size, but hey - it's just an example. – Nils Pipenbrinck Oct 29 '08 at 8:54

58 Volatile is also needed in threaded code when you are playing with data that isn't concurrency protected. And yes there are valid times to be doing that, you can for example write a thread safe circular message queue without needing explicit concurrency protection, but it will need volatiles. – Gordon Wrigley Oct 29 '08 at 9:57

11 Read the C specification harder. Volatile only has defined behavior on memory-mapped device I/O or memory touched by an asynchronous interrupting function. It says *nothing* about threading, and a compiler which optimizes away access to memory touched by multiple threads is conformant. – ephemient Oct 29 '08 at 14:36

10 Do you have a reference for that claim? Every reference I've seen says volatile tells the compiler not to optimize memory accesses on whatever. I've never seen a restriction on whether the memory is RAM or IO. Further on many platforms the compiler can't distinguish between RAM and memory mapped IO. – Gordon Wrigley Oct 29 '08 at 22:28

Another use for `volatile` is signal handlers. If you have code like this:

```
quit = 0;
while (!quit)
{
    /* very small loop which is completely visible to the compiler */
}
```

The compiler is allowed to notice the loop body does not touch the `quit` variable and convert the loop to a `while (true)` loop. Even if the `quit` variable is set on the signal handler for `SIGINT` and `SIGTERM`; the compiler has no way to know that.

However, if the `quit` variable is declared `volatile`, the compiler is forced to load it every time, because it can be modified elsewhere. This is exactly what you want in this situation.

answered Oct 29 '08 at 10:52

CesarB
29.6k ● 4 ● 51 ● 74

when you say "the compiler is forced to load it every time, is it like when compiler decide to optimize a certain variable and we don't declare the variable as volatile, at run time that certain variable is loaded to CPU registers not in memory? – Amit Singh Tomar Mar 17 '15 at 16:34

1 @AmitSinghTomar It means what it says: Every time the code checks the value, it is reloaded. Otherwise, the compiler is allowed to assume that functions that don't take a reference to the variable can't modify it, so assuming as CesarB intended that the above loop does not set `quit`, the compiler can optimise it into a constant loop, assuming that there's no way for `quit` to be changed between iterations. N.B.: This isn't necessarily a good substitute for actual threadsafe programming. – underscore_d Jul 2 '16 at 14:46

if quit is a global variable, then the compiler shall not optimize the while loop, correct ? – Pierre G. Aug 20 '16 at 13:47

1 @PierreG. No, the compiler can always assume that the code is single-threaded, unless told otherwise. That is, in the absence of `volatile` or other markers, it will assume that nothing outside the loop modifies that variable once it enters the loop, even if it's a global variable. – CesarB Sep 12 '16 at 12:37

1 @PierreG. Yes, try for instance compiling `extern int global; void fn(void) { while (global != 0) { } }` with `gcc -O3 -S` and look at the resulting assembly file, on my machine it does `movl global(%rip), %eax`; `testl %eax, %eax`; `je .L1`; `.L4: jmp .L4`, that is, an infinite loop if the global is not zero. Then try adding `volatile` and see the difference. – CesarB Sep 12 '16 at 13:00

`volatile` in C actually came into existence for the purpose of not caching the values of the variable automatically. It will tell the machine not to cache the value of this variable. So it will take the value of the given `volatile` variable from the main memory every time it encounters it. This mechanism is used because at any time the value can be modified by the OS or any interrupt. So using `volatile` will help us accessing the value afresh every time.

edited Apr 4 '16 at 15:59

rphv
3,622 ● 2 ● 17 ● 36

answered Oct 29 '08 at 8:44

Manoj Doubts
4,889 ● 13 ● 35 ● 42

8 simple and best explanation. – nikk Feb 20 '15 at 19:15

Came into existence? Wasn't ´volatile` originally borrowed from C++? Well, I seem to remember... – syntaxerror Jan 25 '16 at 4:51

This is not volatile all about - it also prohibit some reordering if specified as volatile.. – FaceBro Jun 7 '16 at 14:25

`volatile` tells the compiler that your variable may be changed by other means, than the code that is accessing it. e.g., it may be a I/O-mapped memory location. If this is not specified in such cases, some variable accesses can be optimised, e.g., its contents can be held in a register, and the memory location not read back in again.

answered Oct 29 '08 at 8:41

Chris Jester-Young
171k ● 33 ● 319 ● 385

See this article by Andrei Alexandrescu, "volatile - Multithreaded Programmer's Best Friend"

> The **volatile** keyword was devised to prevent compiler optimizations that might render code incorrect in the presence of certain asynchronous events. For example, if you declare a primitive variable as **volatile**, the compiler is not permitted to cache it in a register -- a common optimization that would be disastrous if that variable were shared among multiple threads. So the general rule is, if you have variables of primitive type that must be shared among multiple threads, declare those variables **volatile**. But you can actually do a lot more with this keyword: you can use it to catch code that is not thread safe, and you can do so at compile time. This article shows how it is done; the solution involves a simple smart pointer that also makes it easy to serialize critical sections of code.

The article applies to both `C` and `C++`.

Also see the article "C++ and the Perils of Double-Checked Locking" by Scott Meyers and Andrei Alexandrescu:

> So when dealing with some memory locations (e.g. memory mapped ports or memory referenced by ISRs [Interrupt Service Routines]), some optimizations must be suspended. volatile exists for specifying special treatment for such locations, specifically: (1) the content of a volatile variable is "unstable" (can change by means unknown to the compiler), (2) all writes to volatile data are "observable" so they must be executed religiously, and (3) all operations on volatile data are executed in the sequence in which they appear in the source code. The first two rules ensure proper reading and writing. The last one allows implementation of I/O protocols that mix input and output. This is informally what C and C++'s volatile guarantees.

edited Jul 27 '10 at 14:56

answered Jul 22 '10 at 12:33

Robert S. Barnes
25.8k ● 23 ● 111 ● 161

Does the standard specify whether a read is considered 'observable behavior' if the value is never used? My impression is that it should be, but when I claimed it was elsewhere someone challenged me for a citation. It seems to me that on any platform where a read of a volatile variable could conceivably have any effect, a compiler should be required generate code that performs every indicated read precisely once; without that requirement, it would be difficult to write code which generated a predictable sequence of reads. – supercat Oct 13 '10 at 22:50

@supercat: According to the first article, "If you use the volatile modifier on a variable, the compiler won't cache that variable in registers — each access will hit the actual memory location of that variable." Also, in section §6.7.3.6 of the c99 standard it says: "An object that has volatile-qualified type may be modified in ways unknown to the implementation or have other unknown side effects." It further implies that volatile variables may not be cached in registers and that all reads and writes must be executed in order relative to sequence points, that they are in fact observable. – Robert S. Barnes Oct 14 '10 at 8:28

The latter article indeed states explicitly that reads are side-effects. The former indicates that reads cannot be performed out of sequence, but did not seem to preclude the possibility of them being elided altogether. – supercat Oct 14 '10 at 20:31

"the compiler is not permitted to cache it in a register" - Most RISC arcitectures ae register-machines, so any read-modify-write **has** to cache the object in a registers. `volatile` does not guarantee atomicity. – Olaf May 22 '17 at 17:43

My simple explanation is:

In some scenarios, based on the logic or code, the compiler will do optimisation of variables which it thinks do not change. The `volatile` keyword prevents a variable being optimised.

For example:

```
bool usb_interface_flag = 0;
while(usb_interface_flag == 0)
{
    // execute logic for the scenario where the USB isn't connected
}
```

From the above code, the compiler may think `usb_interface_flag` is defined as 0, and that in the while loop it will be zero forever. After optimisation, the compiler will treat it as `while(true)` all the time, resulting in an infinite loop.

To avoid these kinds of scenarios, we declare the flag as volatile, we are telling to compiler that this value may be changed by an external interface or other module of program, i.e., please don't optimise it. That's the use case for volatile.

edited Apr 20 '15 at 13:23

PROGRAM_IX
326 ● 3 ● 18

answered Jan 24 '15 at 7:01

Venkatakrishna Kalepalli
261 ● 2 ● 6

A marginal use for volatile is the following. Say you want to compute the numerical derivative of a function `f` :

```
double der_f(double x)
{
    static const double h = 1e-3;
    return (f(x + h) - f(x)) / h;
}
```

The problem is that `x+h-x` is generally not equal to `h` due to roundoff errors. Think about it : when you substract very close numbers, you lose a lot of significant digits which can ruin the computation of the derivative (think 1.00001 - 1). A possible workaround could be

```
double der_f2(double x)
{
    static const double h = 1e-3;
    double hh = x + h - x;
    return (f(x + hh) - f(x)) / hh;
}
```

but depending on your platform and compiler switches, the second line of that function may be wiped out by a aggressively optimizing compiler. So you write instead

```
    volatile double hh = x + h;
    hh -= x;
```

to force the compiler to read the memory location containing hh, forfeiting an eventual optimization opportunity.

edited Jun 1 '14 at 9:40

answered Jun 30 '10 at 11:34

Alexandre C.
41.6k ● 7 ● 94 ● 174

What is a difference between using `h` or `hh` in derivative formula? When `hh` is computed the last formula uses it like the first one, with no difference. Maybe it should be `(f(x+h) - f(x))/hh` ? – Sergey Zhukov Sep 22 '14 at 6:02

1 The difference between `h` and `hh` is that `hh` is truncated to some negative power of two by the operation `x + h - x` . In this case, `x + hh` and `x` differ exactly by `hh` . You can also take your formula, it will give the same result, since `x + h` and `x + hh` are equal (it is the denominator which is important here). – Alexandre C. Sep 22 '14 at 18:16

3 Isn't more readable way to write this would be `x1=x+h; d = (f(x1)-f(x))/(x1-x)` ? without using the volatile. – Sergey Zhukov Sep 24 '14 at 19:19

There are two uses. These are specially used more often in embedded development.

1. Compiler will not optimise the functions that uses variables that are defined with volatile keyword
2. Volatile is used to access exact memory locations in RAM, ROM, etc... This is used more often to control memory-mapped devices, access CPU registers and locate specific memory locations.

See examples with assembly listing. Re: Usage of C "volatile" Keyword in Embedded Development

answered Jul 30 '11 at 4:59

Neo Cambell
91 ● 1 ● 1

"Compiler will not optimise the functions that uses variables that are defined with volatile keyword" - that's plain wrong. – Olaf May 22 '17 at 17:46

I'll mention another scenario where volatiles are important.

Suppose you memory-map a file for faster I/O and that file can change behind the scenes (e.g. the file is not on your local hard drive, but is instead served over the network by another computer).

If you access the memory-mapped file's data through pointers to non-volatile objects (at the source code level), then the code generated by the compiler can fetch the same data multiple times without you being aware of it.

If that data happens to change, your program may become using two or more different versions of the data and get into an inconsistent state. This can lead not only to logically incorrect behavior of the program but also to exploitable security holes in it if it processes untrusted files or files from untrusted locations.

If you care about security, and you should, this is an important scenario to consider.

answered Nov 26 '11 at 10:05

Alexey Frunze
49.7k ● 8 ● 47 ● 114

Volatile is also useful, when you want to force the compiler not to optimize a specific code sequence (e.g. for writing a micro-benchmark).

answered Oct 29 '08 at 8:46

Diomidis Spinellis
13.8k ● 2 ● 35 ● 61

---

volatile means the storage is likely to change at anytime and be changed but something outside the control of the user program. This means that if you reference the variable, the program should always check the physical address (ie a mapped input fifo), and not use it in a cached way.

edited Mar 1 at 5:40

Undo ♦
21.9k ● 20 ● 82 ● 108

answered Jul 9 '11 at 3:14

Structure padding
109 ● 2 ● 1

The Wiki say everything about `volatile`:

- volatile (computer programming)

And the Linux kernel's doc also make a excellent notation about `volatile`:

- Why the "volatile" type class should not be used

edited Aug 30 '15 at 18:42

Mahendra Gunawardena
1,469 ● 5 ● 17 ● 35

answered Sep 5 '12 at 14:54

coanor
1,581 ● 2 ● 20 ● 37

A volatile can be changed from outside the compiled code (for example, a program may map a volatile variable to a memory mapped register.) The compiler won't apply certain optimizations to code that handles a volatile variable - for example, it won't load it into a register without writing it to memory. This is important when dealing with hardware registers.

answered Oct 29 '08 at 8:45

Ori Pessach
5,772 ● 4 ● 30 ● 48

In my opinion, you should not expect too much from `volatile`. To illustrate, look at the example in Nils Pipenbrinck's highly-voted answer.

I would say, his example is not suitable for `volatile`. `volatile` is only used to: *prevent the compiler from making useful and desirable optimizations*. It is nothing about the thread safe, atomic access or even memory order.

In that example:

```
void SendCommand (volatile MyHardwareGadget * gadget, int command, int data)
{
  // wait while the gadget is busy:
  while (gadget->isbusy)
  {
    // do nothing here.
```

```
    }
    // set data first:
    gadget->data    = data;
    // writing the command starts the action:
    gadget->command = command;
  }
```

the `gadget->data = data` before `gadget->command = command` only is only guaranteed in compiled code by compiler. At running time, the processor still possibly reorders the data and command assignment, regarding to the processor architecture. The hardware could get the wrong data(suppose gadget is mapped to hardware I/O). The memory barrier is needed between data and command assignment.

edited May 23 '17 at 12:18
Community ♦
1 ● 1

answered Mar 14 '16 at 15:26
Oliver
29 ● 4

I'd say volatile is used to prevent the compiler from making optimizations that would *normally* be useful and desirable. As written, it sounds like `volatile` is degrading the performance for no reason. As for whether it is sufficient, that will depend upon other aspects of the system that the programmer may know more about than the compiler. On the other hand, if a processor guarantees that an instruction to write to a certain address will flush the CPU cache but a compiler provided no way to flush register-cached variables the CPU knows nothing about, flushing the cache would be useless. – supercat Feb 19 '17 at 1:28

it does not allows compiler to automatic changing values of variables. a volatile variable is for dynamic use.

answered May 21 '10 at 19:23
venu
19

protected by tchrist Sep 5 '12 at 18:36

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?