Understanding “volatile” qualifier in C | Set 2 (Examples)

The volatile keyword is intended to prevent the compiler from applying any optimizations on objects that can change in ways that cannot be determined by the compiler.

Objects declared as volatile are omitted from optimization because their values can be changed by code outside the scope of current code at any time. The system always reads the current value of a volatile object from the memory location rather than keeping its value in temporary register at the point it is requested, even if a previous instruction asked for a value from the same object. So the simple question is, how can value of a variable change in such a way that compiler cannot predict. Consider the following cases for answer to this question.

1) Global variables modified by an interrupt service routine outside the scope: For example, a global variable can represent a data port (usually global pointer referred as memory mapped IO) which will be updated dynamically. The code reading data port must be declared as volatile in order to fetch latest data available at the port. Failing to declare variable as volatile, the compiler will optimize the code in such a way that it will read the port only once and keeps using the same value in a temporary register to speed up the program (speed optimization). In general, an ISR used to update these data port when there is an interrupt due to availability of new data

2) Global variables within a multi-threaded application: There are multiple ways for threads communication, viz, message passing, shared memory, mail boxes, etc. A global variable is weak form of shared memory. When two threads sharing information via global variable, they need to be qualified with volatile. Since threads run asynchronously, any update of global variable due to one thread should be fetched freshly by another consumer thread. Compiler can read the global variable and can place them in temporary variable of current thread context. To nullify the effect of compiler optimizations, such global variables to be qualified as volatile

If we do not use volatile qualifier, the following problems may arise
1) Code may not work as expected when optimization is turned on.
2) Code may not work as expected when interrupts are enabled and used.

Let us see an example to understand how compilers interpret volatile keyword. Consider below code, we are changing value of const object using pointer and we are compiling code without optimization option. Hence compiler won’t do any optimization and will change value of const object.

```
/* Compile code without optimization option */
#include <stdio.h>
int main(void)
{
    const int local = 10;
    int *ptr = (int*) &local;
    printf("Initial value of local : %d \n", local);

    *ptr = 100;

    printf("Modified value of local: %d \n", local);

    return 0;
}
```

Run on IDE

When we compile code with “–save-temps” option of gcc it generates 3 output files

1) preprocessed code (having .i extention)
2) assembly code (having .s extention) and
3) object code (having .o option).

We compile code without optimization, that’s why the size of assembly code will be larger (which is highlighted in red color below).

Output:

```
[narendra@ubuntu]$ gcc volatile.c -o volatile –save-temps
[narendra@ubuntu]$ ./volatile
Initial value of local : 10
Modified value of local: 100
[narendra@ubuntu]$ ls -l volatile.s
-rw-r–r– 1 narendra narendra 731 2016-11-19 16:19 volatile.s
[narendra@ubuntu]$
```

Let us compile same code with optimization option (i.e. -O option). In thr below code, “local” is declared as const (and non-volatile), GCC compiler does optimization and ignores the instructions which try to change value of const object. Hence value of const object remains same.

```
/* Compile code with optimization option */
#include <stdio.h>

int main(void)
{
    const int local = 10;
    int *ptr = (int*) &local;

    printf("Initial value of local : %d \n", local);

    *ptr = 100;

    printf("Modified value of local: %d \n", local);

    return 0;
}
```

Run on IDE

For above code, compiler does optimization, that’s why the size of assembly code will reduce.

Output:

```
[narendra@ubuntu]$ gcc -O3 volatile.c -o volatile –save-temps
[narendra@ubuntu]$ ./volatile
Initial value of local : 10
Modified value of local: 10
```

```
[narendra@ubuntu]$ ls -l volatile.s
-rw-r–r– 1 narendra narendra 626 2016-11-19 16:21 volatile.s
```

Let us declare const object as volatile and compile code with optimization option. Although we compile code with optimization option, value of const object will change, because variable is declared as volatile that means don't do any optimization.

```
/* Compile code with optimization option */
#include <stdio.h>

int main(void)
{
    const volatile int local = 10;
    int *ptr = (int*) &local;

    printf("Initial value of local : %d \n", local);

    *ptr = 100;

    printf("Modified value of local: %d \n", local);

    return 0;
}
```

Run on IDE

Output:

```
[narendra@ubuntu]$ gcc -O3 volatile.c -o volatile –save-temp
[narendra@ubuntu]$ ./volatile
Initial value of local : 10
Modified value of local: 100
[narendra@ubuntu]$ ls -l volatile.s
-rw-r–r– 1 narendra narendra 711 2016-11-19 16:22 volatile.s
[narendra@ubuntu]$
```

The above example may not be a good practical example, the purpose was to explain how compilers interpret volatile keyword. As a practical example, think of touch sensor on mobile phones. The driver abstracting touch sensor will read the location of touch and send it to higher level applications. The driver itself should not modify (const-ness) the read location, and make sure it reads the touch input every time fresh (volatile-ness). Such driver must read the touch sensor input in const volatile manner.

Note : The above codes are compiler specific and may not work on all compilers. The purpose of the examples is to make readers understand the concept.

Related Article :

Understanding "volatile" qualifier in C | Set 1 (Introduction)

Refer following links for more details on volatile keyword:

Volatile: A programmer's best friend

Do not use volatile as a synchronization primitive

This article is compiled by "Narendra Kangralkar" and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Practice Tags : C

Article Tags : C C-Storage Classes and Type Qualifiers

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Const Qualifier in C
Understanding “volatile” qualifier in C | Set 1 (Introduction)
Initialization of static variables in C
Understanding “register” keyword in C
Returned values of printf() and scanf()
Message encryption and decryption using UDP server
Storage of integer and character values in C
How will you print numbers from 1 to 100 without using loop? | Set-2
Common Memory/Pointer Related bug in C Programs
Format specifiers in C

(Login to Rate)

3.1 Average Difficulty : **3.1/5.0**
Based on **67** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Feedback Add Notes

Improve Article

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!