

Volatile: Endgame – come, quando e perché usare il type qualifier volatile in C

Scott Lang/Ant-Man: No, se rispettiamo le regole del viaggio nel tempo: non parlare con noi che troveremo in quel passato, non scommettere su eventi sportivi...

Tony Stark/Iron Man: Ti fermo prima che tu vada avanti, Scott. Mi stai seriamente dicendo che il tuo piano per salvare l'universo è basato su "Ritorno al Futuro"?

Il bel **Avengers: Endgame** termina in maniera magistrale il primo ciclo degli **Avengers Marvel**, ed è un bel film che chiude varie trame e sotto-trame iniziate tempo addietro, ed ha molteplici chiavi di lettura che gli permettono di piacere sia ai Cinefili (come me) sia a chi pensa che un Cinema è “un posto come un altro per ripararsi dalla pioggia” [S.Kubrick, 1987].

E cosa centra tutto questo con la famigerata keyword **volatile**? Beh, forse la presenza di trame e sotto-trame di una lunga storia che ha creato non pochi equivoci sull'argomento. Con questo articolo vorrei creare un *Endgame* sull'uso e disuso di **volatile**, ma ho obiettivi limitati e realistici: perlomeno mi accontenterei di dare qualche utile informazione a chi neppure sa cosa sia e, allo stesso tempo, di non confondere le idee a chi la usa già in maniera efficace. *Speriamo bene...*



...ti spiego: un volatile non necessariamente ha le ali...

E direi di cominciare con le buone notizie:

- La prima buona notizia è che **volatile** si usa veramente col contagocce, perché realmente serve solo nella programmazione di bassissimo livello, quella a stretto contatto con l'Hardware: quindi se non siete dei programmatori *hard-embedded* probabilmente non la userete mai.
- La seconda buona notizia è che l'uso improprio di **volatile** è probabile che non faccia molti danni: nel senso che se la usate (per sbaglio) quando in realtà non vi serve è molto probabile che il codice funzioni ugualmente bene: *in quel caso potrete essere fieri di avere un codice ben funzionante **nonostante** l'uso di **volatile** (sono soddisfazioni...).*

Ma c'è anche una cattiva notizia molto somigliante alla seconda descritta sopra:

- L'uso improprio di **volatile** potrebbe anche fare molti danni: nel senso che se la usate per scopi che non sono quelli previsti dallo standard, è molto probabile che il codice abbia dei malfunzionamenti veramente strani e

difficili da capire, roba da sindrome di *“mal di testa del programmatore”*.

E veniamo al dunque: nel titolo di questo articolo preannunciavo la descrizione del **come**, **quando** e **perché** usare **volatile** in C, per cui ora ci tocca cominciare con:

IL COME:

Ricordiamo che **volatile** è un **type qualifier**, e quindi si usa (solo) per aggiungere una proprietà a un tipo qualsiasi già esistente, quindi **il come** è semplicissimo e, ad esempio, possiamo scrivere:

```
int dummy; // un int normalissimo che si chiama dummy
volatile int vol_dummy; // un int di tipo volatile che si chiama vol_dummy
```

E approfitto l'occasione per ricordare che i *type qualifier* sono solo quattro, e che gli altri sono *const*, *restrict* e *_Atomic* : magari ne parleremo in un prossimo articolo (magari uno per ogni qualificatore). E ora siamo pronti a passare al prossimo punto:

IL QUANDO:

In questo caso è sufficiente ricordare i motivi *“storici”* alla base dell'uso di **volatile**, quattro motivi raggruppati sotto la definizione: *“in tutte situazioni in cui il valore della variabile può cambiare senza azione da parte del codice visibile”*. E vediamoli, questi quattro **quando** (*spoiler: il quarto è sbagliatissimo*):

1. Quando ci si interfaccia con un Hardware che cambia il valore stesso della variabile.
2. Quando c'è un gestore di segnali che potrebbe cambiare il valore della variabile.
3. Quando una variabile cambia tra un *setjmp* e il *longjmp* collegato.
4. Quando c'è un altro *thread* in esecuzione che usa anche

lui la stessa variabile (*ERRORE! Anzi, ORRORE!*)

Ecco, il punto 4 è veramente sbagliato, ma non ve lo spiegherò direttamente, lo vedremo nel prossimo punto, che è:

IL PERCHÉ:

Ma perché scrivendo del codice bisogna informare qualche *entità suprema* che la variabile è volatile? Ma che gliene frega all'*entità suprema* (che è, in questo caso, il **compilatore**) che qualcuno esternamente potrebbe modificare il valore della variabile? Sembrerebbe più un problema di *run-time* che di *compile-time*, eppure... beh, la risposta è semplice: perché funzioni il tutto il compilatore **deve** sviluppare del codice-macchina che sia a conoscenza che quella variabile è speciale, che quella variabile non cambia per le operazioni dirette scritte nel codice ma cambia per alcune ingerenze esterne.

E tutto questo è strettamente collegato alle **ottimizzazioni**: dovete sapere (*beh, immagino che lo sappiate già*) che i compilatori hanno la pessima (*o ottima, dipende dai punti di vista*) abitudine di ottimizzare il codice (***...vedi al proposito questo ottimo articolo...***). Quindi, il compilatore *ha bisogno* di sapere se la variabile è volatile, per evitare di eseguire ottimizzazioni dannose su un codice apparentemente “strano” come può apparire un codice **che ha veramente bisogno** di variabili volatili (l'ottimizzatore del compilatore non può capire tutti i dettagli se non glieli esplicitiamo).

E adesso vi propongo un esempio semplicissimo, che è uno dei tanti possibili (*è veramente, ma veramente, semplificato, ma è solo per rendere l'idea*):

```
int main()
{ int dummy = 0; // per semplificare, ma dovrebbe essere
  esterna while (dummy == 0) { // eseguo qualcosa... ; } return
  0;
}
```

ecco, questo semplice codice verrà trasformato dal compilatore (anche con ottimizzazioni minime, ad esempio usando `-O1` nel caso del `GCC`) in:

```
int main()
{ int dummy = 0; // per semplificare, ma dovrebbe essere
  esterna while (1) { // eseguo qualcosa... ; } return 0;
}
```

Questo perché il test della variabile nel `while` è inutile, già che la condizione è sempre vera.

Ma cosa succede se la mia variabile `dummy` è una abilitazione proveniente da una variabile mappata direttamente nell'Hardware (un input digitale, per esempio)? Succede che il test che sembrava inutile in realtà deve essere sempre fatto, e quindi mio codice non funziona più! E che si fa allora? Si fa questo:

```
int main()
{ volatile int vol_dummy = 0; // per semplificare, ma dovrebbe
  essere esterna while (vol_dummy == 0) { // eseguo qualcosa...
  ; } return 0;
}
```

In questo caso il compilatore è informato che non deve assolutamente ottimizzare il `loop`, perché è basato su una variabile volatile, e se provate a ottenere l'*assembler* dalla compilazione ve ne renderete facilmente conto (*certo, spiegare dettagli della programmazione C usando l'assembler è un po' fuorviante, ma in questo caso ci sta bene*). E questo spiega anche la seconda buona notizia descritta sopra: **se usiamo volatile dove non ce n'è bisogno**, alla fin fine solo stiamo disabilitando le ottimizzazioni per una parte limitata del codice, e questo non provoca malfunzionamenti, ma, al massimo, una riduzione delle prestazioni.

E la allora, perché il punto 4 del **quando** era sbagliato? Questo ci porta alla cattiva notizia descritta in testa all'articolo: **se usiamo volatile dove non bisogna usarla**,

potremmo avere dei problemi, e questo nasce da un vecchio malinteso: il motivo per cui la variabile si intende come *“modificabile esternamente”* è valido solo per i primi tre casi elencati sopra, ma non include le modifiche effettuate da un altro *thread*. Una operazione **thread-safe** su una variabile deve essere **atomica** (usando i costrutti forniti dal linguaggio) oppure deve essere protetta dai soliti metodi di sincronizzazione disponibili: *mutex*, *spinlock*, ecc. Le operazioni su una **volatile** non sono atomiche, e quindi non sono adatte per eseguire sincronizzazione (e questo è specificato anche nei vari standard (C, **POSIX** e altri).

E per oggi penso che possa bastare, e spero di non avere aggiunto confusione a un argomento già di per sé abbastanza confuso...

Ciao, e al prossimo post!

<https://italiancoders.it/volatile-endgame-come-quando-e-perche-usare-il-type-qualifier-volatile-in-c/>