Il buono, il brutto, il VLA – come usare i Variable Length Array in C – pt.2

Biondo (il buono): *[contando gli uomini di Sentenza] Uno, due, tre, quattro, cinque, sei... Sei, il numero perfetto!*
Sentenza (il cattivo): *Non è tre il numero perfetto?*
Biondo (il buono): *Sì, ma io ho sei colpi qui dentro...*

Dunque, dove eravamo rimasti? Ah si, **nell'ultimo articolo** (*che avete appena riletto, vero?*) avevamo approvato (con riserva) i **Variable Length Array** (**VLA** per gli amici), che sono facili da usare, utili e con ottime prestazioni, ma allora… perché ho detto che sono adatti al ruolo del *cattivo* nel mitico **Il buono, il brutto, il cattivo** del grande **Sergio Leone**?



…non fidatevi del VLA, parola del buono!…
Presto detto: oltre ai (notevoli) *pro* ci sono anche alcuni (pesanti) *contro*. Prima di seguire ricordiamoci sempre che un VLA si alloca dinamicamente nello *stack* come una variabile automatica con *scope* limitato al blocco di codice dove avviene l'allocazione: dopodiché i (principali) possibili problemi sono:

1. la gestione degli errori è problematica, perché non c'è

maniera di sapere se il VLA è stato allocato bene o ha provocato un *stack overflow* (in questo caso provoca effetti simili a quelli di un errore per ricorsione infinita).
2. il size del VLA si decide a *run-time*, quindi il compilatore deve fare dei giochi un po' strani: in base all'implementazione è possibile che una parte (anche importante) dello *stack* di una funzione venga riservato per un VLA, limitando molto la memoria locale disponibile. Quindi lo *stack overflow* è sempre dietro l'angolo.
3. la portabilità del codice va un po' a farsi benedire: il codice diventa molto *compiler-dependent* e, soprattutto, visto che una buona fetta di programmatori **C** scrivono anche codice per *sistemi embedded* (dove lo *stack* è, spesso, limitato) risulta complicato il *porting* di funzioni da applicazioni *normali* ad applicazioni *embedded*. Funzioni che, magari, smetterebbero di funzionare per motivi misteriosi (*beh, neanche tanto misteriosi…*).
4. E, *dulcis in fundo*: forse per i motivi appena elencati (o per altri ancora) da **C11** in avanti i VLA sono opzionali e subordinati al flag **__STDC_NO_VLA__** del compilatore, *e questo è un brutto segno.*

Che fare allora? Meglio non usarli o usarli con le precauzioni del caso, anche perché le alternative non mancano. *E, con questo, abbiamo trovato il Cattivo!*

E adesso ci tocca cercare qualcuno che sia adatto ai ruoli del *buono* e del *brutto*. Ecco, per il *buono* non c'è problema, il candidato ideale è la cara, buona, vecchia **malloc(3)** che è sempre una garanzia ed è uscita molto bene dal test. Sulla **malloc(3)** è inutile dilungarci, è un punto fermo del **C** e **ne abbiamo già parlato abbondantemente qui**.

E il *brutto*? Beh, per cercare uno adatto dovremo, ahimè, addentrarci nel **lato oscuro della forza**, e cioè in

territorio **C++**…

*(…apro una parentesi: non parlo mai di argomenti che non conosco, perché penso che sia stupido farlo. Per fare un esempio: io non capisco niente di moto e, vi assicuro, nessuno ha mai avuto l'onore di sentirmi disquisire sul mondiale di **MotoGP**. Seguo una filosofia che, sfortunatamente, non è seguita da molta gente, è cioè: "mai parlare solo per dare aria alla bocca". Proprio in virtù di questa coerenza penso di avere i titoli per parlare del C++: lo uso in parallelo al mio amato C da circa trenta (!) anni, e, modestia a parte, penso di saperlo usare bene. Quindi ne posso disquisire, nel bene e nel male. Chiudo la parentesi…)*.

Allora, ho ripreso pari pari l'esempio **C** del post precedente e (facendo il minimo sindacale di modifiche) l'ho trasformato in codice **C++**, per poter, così, aggiungere un test nuovo che usa **std::vector** (*questo è un oggetto particolarmente caro ai C++ lovers, che lo usano anche per condire l'insalata*). Per non ripetere tutto il **codice dello scorso post** vi riporto solo il **main()** e la nuova funzione di test aggiunta (il resto è, praticamente, invariato). Vai col codice!

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <vector> #define MYSIZE 1000000 // variabile dummy per evitare lo svuotamento totale delle funzioni usando g++ -O2
int avoid_optimization; // prototipi locali
void runTest(int iterations, void (*funcptr)(int), int size, const char *name);
void testVLA(int size);
void testMallocVLA(int size);
void testStackFLA(int dum);
void testHeapFLA(int dum);
void testVectorVLA(int size); // funzione main()
int main(int argc, char* argv[])
{ // test argomenti if (argc != 2) { // errore: conteggio argomenti errato printf("%s: wrong arguments counts\n",
```

```
argv[0]); printf("usage: %s vla iterations [e.g.: %s 10000]\n", argv[0], argv[0]); return EXIT_FAILURE; } // estrae iterazioni int iterations = atoi(argv[1]); // esegue test runTest(iterations, &testVLA, MYSIZE, "testVLA"); runTest(iterations, &testMallocVLA, MYSIZE, "testMallocVLA"); runTest(iterations, &testStackFLA, 0, "testStackFLA"); runTest(iterations, &testHeapFLA, 0, "testHeapFLA"); runTest(iterations, &testVectorVLA, MYSIZE, "testVectorVLA"); // esce return EXIT_SUCCESS;
} // testVectorVLA() - funzione per eseguire il test del vector VLA
void testVectorVLA( int size) // size per std::vector
{ std::vector<int> vectorvla(size); // loop di test for (int i = 0; i < size; i++) vectorvla[i] = i; // istruzione per evitare lo svuotamento totale della funzione usando g++ -O2 avoid_optimization = vectorvla[size / 2];
}
```

Compilando (con/senza ottimizzazioni) ed eseguendo questo codice i risultati sono i seguenti:

```
aldo@Linux $ g++ -O0 vlacpp.cpp -o vlacpp
aldo@Linux $ ./vlacpp 2000
testVLA - Tempo trascorso: 3.912879 secondi
testMallocVLA - Tempo trascorso: 2.727576 secondi
testStackFLA - Tempo trascorso: 3.662505 secondi
testHeapFLA - Tempo trascorso: 3.625115 secondi
testVectorVLA - Tempo trascorso: 8.964794 secondi aldo@Linux $ g++ -O2 vlacpp.cpp -o vlacpp
aldo@Linux $ ./vlacpp 2000
testVLA - Tempo trascorso: 0.630394 secondi
testMallocVLA - Tempo trascorso: 0.615095 secondi
testStackFLA - Tempo trascorso: 0.218578 secondi
testHeapFLA - Tempo trascorso: 0.208932 secondi
testVectorVLA - Tempo trascorso: 0.773074 secondi
```

E, allora, come si è comportato *std::vector* ? *direi che i numeri parlano da soli, quindi abbiamo un buon candidato per il ruolo del brutto*… ma vabbè, mettiamola giù in maniera diplomatica: diciamo che abbiamo due notizie, una buona e una cattiva:

- la buona notizia è che il **C++** è efficiente come il **C** (e su questo non avevo dubbi), infatti il nostro programma **C** trasformato in **C++** ottiene (nei primi quattro test) le stesse prestazioni (**andate a controllare là**, se non ci credete, eh!).
- la brutta notizia è che il **C++** è efficiente come il **C**, *ma solo se lo usate come il **C**, quindi niente **STL** e ammennicoli vari.* Ma, comunque, con l'ottimizzazione attivata, anche *std::vector funzionicchia…*

(…apro un altra parentesi: ovviamente la brutta notizia qui sopra non deriva solo dal semplice test proposto in questo post: deriva da anni ed anni di osservazioni ed uso intensivo di entrambi i linguaggi, ci mancherebbe solo. Chiudo la parentesi…).

E, visto quanto sopra, credo che sia il caso di esporre (brevemente) una mia opinione: il **C++** è un grande linguaggio potente, efficiente ed espressivo (*è parente stretto del C!*), con cui si può scrivere del Software di alta qualità. Ma i risultati migliori (perlomeno in termini di prestazioni e fluidità del codice) si ottengono usandolo per quello che era stato concepito originalmente, e cioè come un semplice **C a oggetti**. La piega che ha preso in seguito *(da quando è caduto nelle mani dei malefici committee ISO*) non mi piace e non mi convince… ma, fortunatamente (e questo è importante), continua a poter essere usato nella sua essenza, quella che permette di scrivere a oggetti usando un linguaggio (quasi) identico al **C** (*e questo si aggancia alla buona notizia qui sopra… e do per scontato che prestazioni e fluidità del codice siano estremamente importanti, eh!*).

Ah, una ultima precisazione per chi si è sorpreso del codice **C++** (qui sopra) che include un VLA: è una gentile offerta dal nostro amato GCC (nella sua incarnazione *g++*). Quindi è un estensione del linguaggio fornita dal compilatore, visto che i VLA non fanno parte del **C++** standard (almeno fino a **C++14**).

Nel prossimo post, per chiudere il cerchio, parleremo di un parente stretto dei VLA, e cioè della funzione **alloca(3)**. *Sarà un altro buono, un altro brutto o un altro cattivo?*

Ciao e al prossimo post!

https://italiancoders.it/il-buono-il-brutto-il-vla-come-usare-i-variable-length-array-in-c-pt-2/