Cry Keyword: complex – come usare le nuove keyword del C – pt.1

Michael "Mike" Milo: *[riferendosi ai federali che lo hanno appena lasciato andare in cambio di una mazzetta] Se avessero un cervello, sarebbero pericolosi...*

Celebriamo oggi l'ultimo film che ci ha regalato un mito vivente del **Cinema** con la **C** maiuscola, **Clint Eastwood**. **Cry Macho** è un road-movie *"crepuscolare"* in cui il nostro Clint interpreta un vecchio cow-boy impegnato in un viaggio, malinconico e gioioso allo stesso tempo, che parte dal sud degli US e si prolunga nel Messico. E questo ci può ispirare a iniziare anche noi un viaggio alla scoperta delle ultime novità del **C**, quelle che pochi usano ma sono, come sempre, molto utili: nel **C** si aggiungono poche cose ma succose (ad esempio **complex**, di cui parleremo tra poco).



…ah, se esistessero i numeri immaginari. Peccato che sono solo

frutto della mia immaginazione…
E comincerò con una auto-citazione dal **mio ultimo articolo** (*dopodiché mi auto-denuncerò per plagio*):

…un po' come è successo, per esempio, con le variabili booleane _Bool definite nell'header stdbool.h: questa norma del nome che inizia con undescore+maiuscola è seguita per tutte le nuove keyword del C, che possono, poi, essere usate con l'header corrispondente per "raddrizzare" il nome…

Ecco, credo che, per mantenere la linea con gli ultimi articoli sia venuto il momento di parlare delle nuove (*vabbè, risalgono al C99 e al C11, diciamo allora "relativamente nuove"*) *keyword* del C, quelle che sono state introdotte un po' per volta, col contagocce (*e giustamente! Perché il C è un linguaggio solido, stabile e senza fronzoli)*. E visto che sono arrivate quando il linguaggio era già da qualche annetto che veniva usato in lungo e in largo per il mondo (è da sempre uno dei linguaggi più usati) bisognava avere un occhio di riguardo per il codice già scritto con implementazioni private delle *keyword* non ancora standardizzate.

E *bool* è il classico esempio: supponendo che qualcuno (in molti, probabilmente) abbia scritto codice con un tipo booleano *user-defined* e lo abbia chiamato *"bool"* (*molto originale, no?*), l'introduzione nel linguaggio di un nuovo tipo con lo stesso nome (e quindi riconosciuto da compilatore) avrebbe creato qualche conflitto con il codice pre-esistente, quindi il committee ISO ha pensato bene che tutte le nuove *keyword* introdotte si sarebbero chiamate con nomi che iniziano con *undescore+maiuscola*, cominciando con **_Bool** che, credo sia stato il primo esempio (è stato introdotto nel C99).

Come funziona questo? È semplicissimo: se voglio usare il nuovo tipo booleano scrivo:

```
#include <stdbool.h> ... bool mybool_a = true; bool mybool_b = false; ...
```

Evidentemente dentro l'*header* standard *stdbool.h* vengono definite, tra le altre cose, le macro *true* (che vale 1), *false* (che vale 0) e *bool* (che corrisponde al nuovo tipo standard **_Bool**). E invece, se ho una mia vecchia implementazione di *bool* e voglio continuare a usarla basta non includere *stdbool.h* e amici come prima.

E quali sono le *"relativamente nuove" keyword* del C? Eccole!

C99:
_Bool
_Complex
_Imaginary C11:
_Alignas
_Alignof
_Atomic
_Generic
_Noreturn
_Static_assert
_Thread_local

di alcune ho già parlato **recentemente** (**_Atomic** nello scorso articolo e **_Bool** proprio ora, non so se vene siete accorti) o **anticamente** (**_Thread_local**, usando però **__thread**, che è una estensione del GCC sostanzialmente identica). In questo articolo direi di soffermarci sulle due che ci mancano del C99, ovvero **_Complex** e **_Imaginary**.

_Complex e **_Imaginary**, come si può ben intuire dai nomi, sono due nuovi *type specifier (tipo char, int, ecc.)* che permettono l'uso dei **numeri complessi** nel **C**. *Nessuno storca il naso* (*"ma chi ha bisogno dei numeri complessi?"*): non sono di uso molto comune, ma non guasta poterli usare, e se non vi servono pensate che chi li usa (e potrebbero essere più di quelli che pensate) vi manderà maledizioni se dite che se ne può fare a meno (*quell'irascibile di **mio cuggino** ad esempio li usa spesso*). Ovviamente non è questa la sede per descrivere cosa sono e a cosa servono i numeri complessi, visto che ci sono milioni di fonti che li descrivono molto meglio di quanto

potrei fare io.

Queste due nuove *keyword* viaggiano in coppia, e si possono utilizzare, entrambe, attraverso l'*header complex.h*. Un solo *header* per due nuove *keyword*? Ebbene si, sono troppo collegate per usarne due diversi, infatti **_Imaginary** serve a definire la parte immaginaria di un numero complesso, quindi è giustificato che tutto sia definito in *complex.h*. E come si usano? Prima di tutto bisogna evidenziare che, per la loro natura (matematica) stessa, si accoppiano a tipi *float*, e che essendo *type specifier* c'è una certa libertà di scrittura, per cui possiamo scrivere questo:

```
... ... // questo formato è Ok _Complex float cf1; _Complex double cd1; _Complex long double cld1; // ma anche questo è Ok float _Complex cf2; double _Complex cd2; long double _Complex cld2; ...
```

e, usando le macro definite nell'apposito *header complex.h*, possiamo *"abbellire"* il codice in:

```
#include <complex.h> ... // questo formato è Ok complex float cf1; complex double cd1; complex long double cld1; // ma anche questo è Ok float complex cf2; double complex cd2; long double complex cld2; ...
```

E adesso è venuto il momento di un bel caso pratico di uso di **_Complex**, che è, spero, più chiaro di mille spiegazioni. Facciamo cantare il codice!

```
#include <stdio.h>
#include <complex.h> int main(void)
{ // dichiaro un numero complesso complex float cf = 5 + 3 * I; // eseguo due operazioni aritmetiche complex float mult_cf = cf * 2; complex float div_cf = mult_cf / 2; // stampo i risultati printf("(5.0 + 3.0i) * 2 = %.1f %+.1fi\n", creal(mult_cf), cimag(mult_cf)); printf("(%.1f %+.1fi) / 2 = %.1f %+.1fi\n", creal(mult_cf), cimag(mult_cf), creal(div_cf), cimag(div_cf));
}
```

credo che l'esempio sia abbastanza chiaro: dichiariamo un numero complesso, lo moltiplichiamo per 2 e dividiamo il risultato per 2, per ottenere lo stesso valore di partenza (tanto per dimostrare la correttezza delle operazioni). Se compilate ed eseguite otterrete:

```
aldo@mylinux:~/blogtest$ ./trycomplex
(5.0 + 3.0i) * 2 = 10.0 +6.0i
(10.0 +6.0i) / 2 = 5.0 +3.0i
```

Funziona! Notare, nel codice, la presenza della macro **I** (definita, ovviamente, in *complex.h*) che serve a definire la parte immaginaria del numero complesso (si usa come moltiplicatore per trasformare un *"numero normale"* in *"numero immaginario"*).

Notare anche che per stampare i valori ho usato delle funzioni di libreria i cui prototipi sono dichiarati in *complex.h*: **creal()** e **cimag()** che, come si intuisce dai nomi, servono a estrarre, rispettivamente, le parti reale e immaginarie di un numero complesso, che poi possono essere agevolmente stampate con una **printf()**, come nell'esempio. Nel nostro *header* sono dichiarate molte altre di queste funzioni per numeri complessi, con nomi facilmente riconducibili alle equivalenti funzioni per *"numeri normali"*. Abbiamo, ad esempio, **cabs()** per il valore assoluto, **clog()** per il logaritmo naturale, **ccos()** per il coseno, e chi più ne ha più ne metta.

Ok, per oggi può bastare. Nei prossimi articoli parleremo delle altre *keyword* della lista mostrata all'inizio dell'articolo. E mi raccomando: non trattenete il respiro nell'attesa!

Ciao, e al prossimo post!

https://italiancoders.it/cry-keyword-complex-come-usare-le-nuove-keyword-del-c-pt-1/