

## Calc-verifier

Creare un programma **calc-verifier.c** in linguaggio C che accetti invocazioni sulla riga di comando del tipo:

```
calc-verifier <calc-file-1> <calc-file-2> ... <calc-file- n >
```

Il programma dovrà essere in grado di verificare la correttezza di una sequenza di semplici operazioni matematiche descritte nei file testuali forniti in input.

Ogni file contiene: - nella prima riga il valore iniziale del calcolo; - una serie di righe del tipo `valore operatore valore`, dove operazione può essere una tra `( + , - , x )` e valore un valore intero da usare come secondo operando insieme al valore ottenuto dall'operazione precedente; - una ultima riga contenente il valore finale atteso. Alcuni esempi sono forniti a corredo: `calc1.txt`, `calc2.txt`, `calc3.txt`.

Al suo avvio il programma creerà  $n+3$  thread:

- $n$  thread CALC-i che si occuperanno di leggere il rispettivo file in input e di coordinare il calcolo/verifica;
- tre thread ADD, SUB e MUL che avranno il compito di supportare gli altri thread per applicare la rispettiva operazione quando necessario.

I thread si coordineranno tramite variabili condizione e mutex: da usare in numero e modalità opportune da determinare da parte dello studente. Le uniche strutture dati condivise dai thread (oltre agli strumenti di coordinamento predetti) saranno:

- `operando_1`: intero di tipo `long long`;
- `operando_2`: intero di tipo `long long`;
- `risultato`: intero di tipo `long long`;
- `operazione`: identificativo dell'operazione da computare;
- `richiedente`: identificativo del thread che ha richiesto l'operazione.

Un thread di tipo CALC dovrà leggere il valore iniziale dalla prima riga e, per ogni operazione incontrata, depositare i valori opportuni nelle variabili condivise e attivare il thread per effettuare il relativo calcolo. Come ultima riga del file (quindi senza segno d'operazione) troverà il risultato atteso: il thread riporterà l'esito del controllo sul proprio standard output.

Il programma dovrà funzionare con un qualunque numero di file in input. Tutti i thread secondari dovranno terminare spontaneamente alla fine dei lavori attesi e alla fine il thread principale dovrà visualizzare un riepilogo sul numero di verifiche andate a buon fine.

Si chiede di rispettare la struttura dell'output riportato nell'esempio a seguire.

**Tempo** : 2 ore

L'output atteso sui file di esempio è il seguente:

```
$ ./calc-verifier calc1.txt calc2.txt
```

```

[CALC-1] file da verificare: 'calc1.txt'
[CALC-2] file da verificare: 'calc2.txt'
[CALC-1] valore iniziale della computazione: 7
[CALC-1] prossima operazione: '+ 30'
[ADD] calcolo effettuato:  $7 + 30 = 37$ 
[CALC-1] risultato ricevuto: 37
[CALC-2] valore iniziale della computazione: 3
[CALC-2] prossima operazione: '+ 6'
[ADD] calcolo effettuato:  $3 + 6 = 9$ 
[CALC-2] risultato ricevuto: 9
[CALC-1] prossima operazione: '+ 4'
[ADD] calcolo effettuato:  $37 + 4 = 41$ 
[CALC-1] risultato ricevuto: 41
[CALC-2] prossima operazione: '- 5'
[SUB] calcolo effettuato:  $9 - 5 = 4$ 
[CALC-2] risultato ricevuto: 4
[CALC-1] prossima operazione: '+ 3'
[ADD] calcolo effettuato:  $41 + 3 = 44$ 
[CALC-1] risultato ricevuto: 44
[CALC-1] prossima operazione: 'x 6'
[MUL] calcolo effettuato:  $44 \times 6 = 264$ 
[CALC-1] risultato ricevuto: 264

...

[CALC-2] prossima operazione: 'x 3'
[MUL] calcolo effettuato:  $-1704708437975 \times 3 = -$ 
[CALC-2] risultato ricevuto: -
[CALC-2] computazione terminata in modo corretto: -
[CALC-1] prossima operazione: '- 2'
[SUB] calcolo effettuato:  $3512341814982522 - 6 = 3512341814982520$ 
[CALC-1] risultato ricevuto: 3512341814982520
[CALC-1] computazione terminata in modo corretto: 3512341814982520
[MAIN] verifiche completate con successo: 2/

```