Riferimenti

Per quanto non specificato in queste indicazioni si rimanda agli altri documenti relativi a questo progetto:

- Note generali
- Presentazione del progetto e del verificatore
- Annunci all'interno del verificatore
- Annunci sui canali Telegram del tutor di riferimento

Chi contattare in caso di bisogno

Domande burocratiche

Contattate il professore di riferimento del vostro scaglione

Domande di carattere generale sul progetto

Contattare il proprio tutor di riferimento tramite email:

- Cognomi da A a CORTI:
  Emanuel Alogna emanuel.alogna@mail.polimi.it

- Cognomi da COSENTINO a LO BUE:
  Valentina Deda deda.valentina@mail.polimi.it

- Cognomi da LO GRASSO a RADAELLI JACOPO:
  Andrea Miotto andrea1.miotto@mail.polimi.it

- Cognomi da RADAELLI SIMONE a ZOCCHE:
  Luca Napoletano luca.napoletano@mail.polimi.it

Per abbreviare il più possibile i tempi di risposta, inserire nell'oggetto della email [ProvaFinaleAPI].

Domande specifiche sulla propria implementazione

Per questo genere di domande sono previsti 5 incontri di tutorato tra la sessione estiva e quella settembre.
Per queste problematiche vi preghiamo di non contattarci per email in quanto una risposta esaustiva deve tener conto di fattori difficilmente valutabili non in presenza.
In generale vi preghiamo quindi di non inviarci per email il vostro codice sorgente.

Ambiente di sviluppo

Non ci sono vincoli che limitino la scelta dell'ambiente di sviluppo da utilizzare, tuttavia consigliamo una qualsiasi distribuzione Linux per emulare al meglio il verificatore (suggeriamo Ubuntu per la semplicità di utilizzo).
Riteniamo che gli strumenti mostrati durante il primo tutorato - e riportati nell'apposita sezione di questo documento - siano ideali per guidarvi nelle operazioni di debug e miglioramento del vostro progetto.
L'uso di questi strumenti e, in generale, un attento debugging devono essere operazioni preliminari a qualsiasi richiesta di assistenza ai tutor.

Deadline

Per i laureandi di luglio la scadenza è alle ore 23:59 del giorno 10 luglio.
Per tutti gli altri la scadenza è alle ore 23:59 del giorno 12 settembre.
Non sono previsti recuperi.

Specifiche e assunzioni

Input

L'input è fornito da stdin.
Si assuma che la sequenza dei comandi di input sia sintatticamente corretta.
I comandi da leggere sono di 6 tipi, con i formati elencati di seguito:

- `addent ⟨id_ent⟩`
- `delent ⟨id_ent⟩`
- `addrel ⟨id_orig⟩ ⟨id_dest⟩ ⟨id_rel⟩`
- `delrel ⟨id_orig⟩ ⟨id_dest⟩ ⟨id_rel⟩`
- `report`
- `end`

Output

L'output deve essere fornito su stdout.

L'unico comando letto da input che produce un output è il comando `report`.
La sequenza deve essere prodotta nel modo seguente:

⟨id_rel1⟩ ⟨id_ent1⟩ ⟨n_rel1⟩; ⟨id_rel2⟩ ⟨id_ent2⟩ ⟨n_rel2⟩; …

- Le relazioni in output sono ordinate in ordine crescente di identificativo
- Se per un tipo di relazione ci sono più entità che sono riceventi del numero massimo di relazioni, queste vengono prodotte in ordine crescente di identificativo. Per esempio:
  ⟨id_rel1⟩ ⟨id_ent1_1⟩ ⟨id_ent1_2⟩ ⟨id_ent1_3⟩ … ⟨n_rel1⟩;
- Se vengono rimosse tutte le relazioni con un certo identificatore, esso non compare nei successivi output del comando report
- Se non ci sono relazioni tra le entità, l'output sarà “none” (senza virgolette)
- L’ordinamento degli identificativi segue la tabella dei caratteri ASCII, per cui vale il seguente ordine: - < 1 < A < _ < a
- Le varie parti di ogni comando e di ogni sequenza di output sono separate da spazi

Altre specifiche

- Sono ammesse entità in relazione con sé stesse
- E’ possibile avere in input dei comandi (diversi da addent) riguardanti entità inesistenti. In questo caso, il programma dovrà semplicemente ignorarli

Verificatore (dum-e.deib.polimi.it)

Task, punteggio e voto

Il task di tutorial non contribuisce in alcun modo all’esito della prova.

Sono previsti 6 task. Ognuno di questi si compone di:
- 1 subtask pubblico - ovvero con input e output atteso pubblici (0 punti)
- 3 subtask privati di difficoltà crescente (rispettivamente 3 + 1 + 1 punti)

Il punteggio massimo ottenibile è 30 punti + 1 punto extra per la lode ottenibile superando un apposito task.

Per superare l’esame è necessario:
- Superare almeno tutti i subtask pubblici e il subtask privato più semplice di ogni task (quello che vale 3 punti)
- Non è sufficiente ottenere 18 punti se il vincolo di cui sopra non è soddisfatto
- Verrà valutata l’ultima implementazione sottomessa in ogni task
- L’implementazione valutata deve essere identica per tutti i task

Annullamento della prova

Verranno annullati tutti i progetti coinvolti in caso di:

- Copia (anche parziale)
- Distribuzione (con qualsiasi mezzo)
- Tentativi di manomissione della piattaforma di valutazione

Debugging

I comandi presenti in questa sezione devono essere eseguiti da terminale in ambiente Unix.
Utilizzando, come consigliato, un sistema Linux, i comandi saranno accettati senza problemi.

Pipelining

Di seguito due comandi utili per testare la propria implementazione, basandosi sui subtask pubblici (input e relativo output atteso)

- `cat input.txt | ./eseguibile > output.txt`
- `diff output.txt output_pubblico.txt`

Software e comandi utili

GCC (compilatore)

Consigliamo di compilare il vostro programma con il comando:

```
gcc sorgente.c -Wmaybe-uninitialized -Wuninitialized -Wall -
pedantic -Werror -g3
```

Dalla suite di Valgrind:

Memcheck: riporta memory leak ed uso di variabili non inizializzate

```
valgrind ./eseguibile
```

consigliamo le opzioni `--leak-check=full` e `--track=origins=yes`

Callgrind + Kcachegrind: callgrind produce un rapporto testuale del tempo di esecuzione delle funzioni chiamate in un programma. kcachegrind prende l'output generato da callgrind e visualizza i risultati ottenuti su interfaccia grafica

```
1. valgrind --tool=callgrind ./eseguibile
2. kcachegrind callgrind.out.xxxx
```

Massif + Massif-visualizer: massif calcola l'utilizzo di memoria (heap only). Come visualizzatore consigliamo massif-visualizer.

```
1. valgrind --tool=massif ./eseguibile
2. massif-visualizer massif.out.xxxx
```

Address Sanitizer: controlla accessi fuori dai vincoli di dimensione sia su heap che su stack. La segnalazione è precisa al byte e vi indica anche in quale frame è stata allocata la variabile locale.

Compilate aggiungendo alle opzioni di gcc le due seguenti:
`-fsanitize=address -lasan`

ed eseguite il binario ottenuto.

Nel caso la libreria di runtime non sia caricata, forzate il pre caricamento passando il percorso alla variabile d'ambiente LD_PRELOAD.
Esempio con un programma che si chiama test:

```
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libasan.so.5 ./test
```

GDB (debugger)

- gdb eseguibile
- run [< file_di_input]
- list [funzione | riga]
- break [funzione | riga]
- [x | print | explore] variabile
- [continue | next | step]
- backtrace
- where

Consigliamo di dare un'occhiata a questa guida mono-foglio A4 con i comandi più comuni.