Tutorato
Prova Finale Algoritmi e Strutture Dati
2021-2022

Luisa Cicolini, luisa.cicolini@mail.polimi.it
Mauro Famà, mauro.fama@mail.polimi.it

Milano, 09.06.22

Comunicazioni

- **Gruppo telegram**: https://t.me/+rCF8i8-Qs6IjZWM8

- Per comunicare con noi - e noi con voi :)

- Domande **chiare** e **ben poste** - possibilmente con screenshot/errori completi

Scadenze

- Per i laureandi di **luglio**
  4 luglio, ore 23.59. **Segnalate** via email al docente la necessità di valutazione
- Per tutti gli altri
  9 settembre, ore 23.59, dopo di che la piattaforma verrà chiusa (non serve segnalazione via email)
- **NB:** laureandi gennaio/febbraio 2023
  con >= **145 CFU** + **iscrizione all'esame di laurea** → la piattaforma sarà riaperta indicativamente tra il 31 gennaio 2023 e l'11 febbraio 2023, fino alle ore 23.59

Sistema anti-plagio

Il progetto è **individuale**: il progetto sarà annullato in caso di **plagio, distribuzione, manomissione** della piattaforma

➜ Attenzione alle repo pubbliche

➜ Sia a chi copia, sia a chi fa copiare!

Sintesi delle Specifiche: WordChecker

- Sistema per il controllo della corrispondenza tra le lettere di 2 parole di ugual lunghezza

- Parola = sequenza di simboli: a-z, A-Z, 0-9, “-”, “_”

Sintesi delle Specifiche: WordChecker

- Lettura da standard input di una sequenza di informazioni e istruzioni: lunghezza *k*, sequenza di parole lunghe k
- Stringhe in output a seconda dei casi.
- Ipotesi: la sequenza di parole non contenga duplicati
- Lettura da standard input di una sequenza di "partite", in cui l'inizio di ogni nuova partita è marcato dal comando (letto sempre da input) *+nuova_partita*

Ambiente di sviluppo e compilazione

- Ambiente di sviluppo consigliato: **Linux**
- IDE consigliato: **CLion**
- Compilatore: **gcc**

```
$gcc –Wall –Werror –O2 –g3 sorgente.c –o eseguibile
```

- *Wall*: mostra tutti i warnings
- *Werror*: tutti i warnings sono trattati come errori
- *O2*: ottimizzazioni del codice
- *g3*: include le informazioni utili per il debugger

Comandi utili (linux)

- Fornire contenuto del file in input al programma

```
$./programma < public_input_file
```

- Scrivere output del programma su file

```
$./programma < public_input_file > program_output
```

Comandi utili (linux)

- Confrontare contenuto di due file

```
$ diff ./public_output ./program_output
```

➜ nessun output significa che i due file sono identici!

Visualizzatori interattivi: Meld o Kdiff

Consigli utili

Se non sapete cosa fare

È il vostro migliore amico

Il verificatore

https://dum-e.deib.polimi.it/

➜ Come allegato al task **Open** trovate il generatore di test, insieme all'archivio dei test case

Errori - verificatore

- **Output is not correct**: causato da errata implementazione, errato parsing dell'input o errata scrittura dell'output ➜ testare in locale con test pubblico.
- **Execution timed out – execution killed**: l’implementazione non soddisfa i vincoli di tempo e memoria del test ➜ valgrind per identificare le parti di codice più dispendiose o memory leaks; valutare strutture dati appropriate.

Errori - verificatore

- **Execution killed with signal 11**: si è verificato un errore dovuto a violazioni di accesso a VMA (segmentation fault) ➜ utilizzare address sanitizer/valgrind in locale.
- **Execution failed because the return code was nonzero**: il main non ritorna 0 o il programma termina inaspettatamente ➜ debug locale

Tools utili

Command Line

Valgrind

- Suite di Strumenti per l'**analisi dinamica** di programmi
  - Nasce per intercettare errori nella gestione della mem
  - Estesa per analisi di prestazioni:
    - **Memcheck**: controllo errori gestione memoria
    - **Callgrind**: tracciamento tempo macchina per funzione
    - **Massif**: tracciamento quantità di memoria dinamica
- La suite valgrind si basa sul modificare l'eseguibile a **runtime**
  - Introduce un overhead (2x circa)
  - Non è compatibile con address sanitizer (usare alternati)

Valgrind

- Installazione (Ubuntu):

```
$ sudo apt-get install valgrind
```

Valgrind - Memcheck

- Installazione (Ubuntu):

```
$ sudo apt-get install valgrind
```

- Utilizzo: Compilare il sorgente con il flag -g3

```
$ valgrind --leak-check=full --show-leak-kinds=all
--track-origins=yes ./eseguibile
```

Valgrind - Memcheck

- Installazione (Ubuntu):

```
$ sudo apt-get install valgrind
```

- Utilizzo: Compilare il sorgente con il flag -g3

```
$ valgrind --leak-check=full --show-leak-kinds=all
--track-origins=yes ./eseguibile
```

➜ **Memcheck**: rileva errori nell'uso della memoria dinamica:

- Deallocazioni mancanti (memory leaks)
- Use after free
- Doppie free

Valgrind - callgrind

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install kcachegrind
```

Valgrind - callgrind

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install kcachegrind
```

- Generazione del report testuale:

```
$ valgrind --tool=callgrind
--callgrind-out-file=outputfile ./eseguibile
```

Valgrind - callgrind

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install kcachegrind
```

- Generazione del report testuale:

```
$ valgrind --tool=callgrind
--callgrind-out-file=outputfile ./eseguibile
```

- Analisi con kcachegrind:

```
$ kcachegrind outputfile
```

Valgrind - callgrind

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install kcachegrind
```

- Generazione del report testuale:

```
$ valgrind --tool=callgrind
--callgrind-out-file=outputfile ./eseguibile
```

- Analisi con kcachegrind:

```
$ kcachegrind outputfile
```

➜ visualizzare il tempo impiegato nelle diverse funzioni/istruzioni del programma

Valgrind - massif

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install massif-visualizer
```

Valgrind - massif

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install massif-visualizer
```

- Generazione del report testuale:

```
$ valgrind --tool=massif --massif-out-file=outputfile
./eseguibile
```

Valgrind - massif

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install massif-visualizer
```

- Generazione del report testuale:

```
$ valgrind --tool=massif --massif-out-file=outputfile
                                ./eseguibile
```

- Analisi con massif-visualizer:

```
$ massif-visualizer outputfile
```

Valgrind - massif

- Installazione GUI (Ubuntu):

```
$ sudo apt-get install massif-visualizer
```

- Generazione del report testuale:

```
$ valgrind --tool=massif --massif-out-file=outputfile
                                ./eseguibile
```

- Analisi con massif-visualizer:

```
$ massif-visualizer outputfile
```

➜ visualizzare l'andamento della memoria allocata nel corso dell'esecuzione

Address Sanitizer

- Installazione (Ubuntu):

```
$ sudo apt install libasan5
```

Address Sanitizer

- Combinazione di compilatore modificato + libreria a runtime: intercetta **violazioni di *accesso*** a memoria
- Byte-accurate (localizza accessi fuori da limiti anche di un singolo byte)
- Aggiunge un overhead in tempo moderato (1.5x)
- Consiglio: utilizzatelo *sempre* durante lo sviluppo
  - Disabilitatelo solo per usare valgrind

Address Sanitizer

- Installazione (Ubuntu):

```
$ sudo apt install libasan5
```

- Utilizzo: Compilare il sorgente aggiungendo la flag:

```
$ gcc main.c -o main -fsanitize=address
```

Address Sanitizer

- Installazione (Ubuntu):

```
$ sudo apt install libasan5
```

- Utilizzo: Compilare il sorgente aggiungendo la flag:

```
$ gcc main.c –o main -fsanitize=address
```

- Eseguire regolarmente il programma

➔ Vengono intercettati errori di accesso in memoria:

- Accesso oltre i limiti di un array
- Dereferenziazione di null pointers
- Accesso oltre i limiti della memoria allocata

Tools utili

IDE (CLion)

CLion - Download

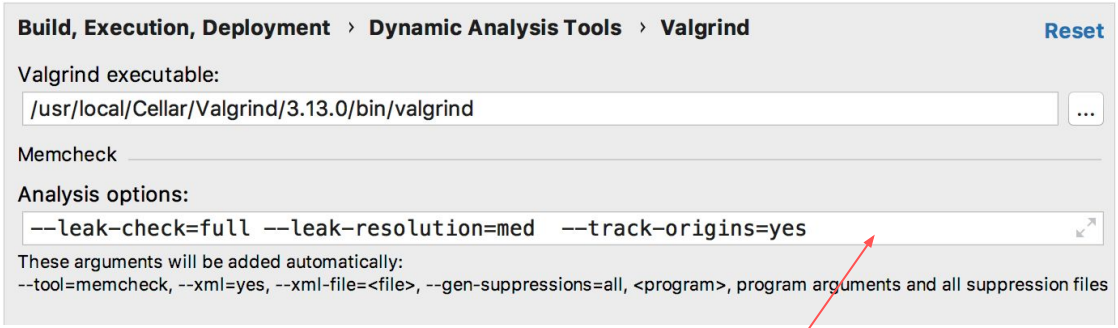
Sarà necessario attivare la licenza studente su JetBrains (basta utilizzare la mail istituzionale): https://www.jetbrains.com/community/education/#students

In seguito potete scaricare l'IDE: https://www.jetbrains.com/clion/download

CLion - Valgrind setup

Installazione: `$ sudo apt-get install valgrind`

In CLion: Settings / Preferences | Build, Execution, Deployment | Dynamic Analysis Tools | Valgrind



--show-leak-kinds=all

https://www.jetbrains.com/help/clion/memory-profiling-with-valgrind.html#start

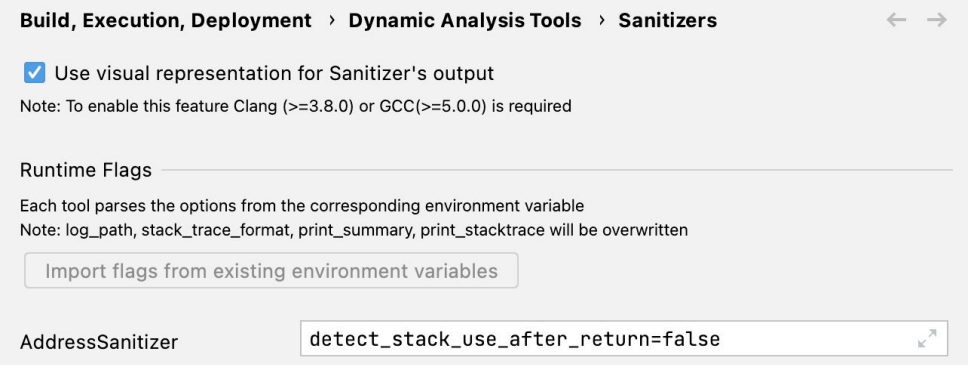
CLion - AddressSanitizer setup

Fa parte di GCC dalla versione 4.8

Aggiungi a **CMakeLists.txt**:

```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -fsanitize=address -static-libasan")
```

In CLion: Settings / Preferences | Build, Execution, Deployment | Dynamic Analysis Tools | Sanitizer



https://www.jetbrains.com/help/clion/google-sanitizers.html

Grazie per l’attenzione, ci sentiamo su telegram :)