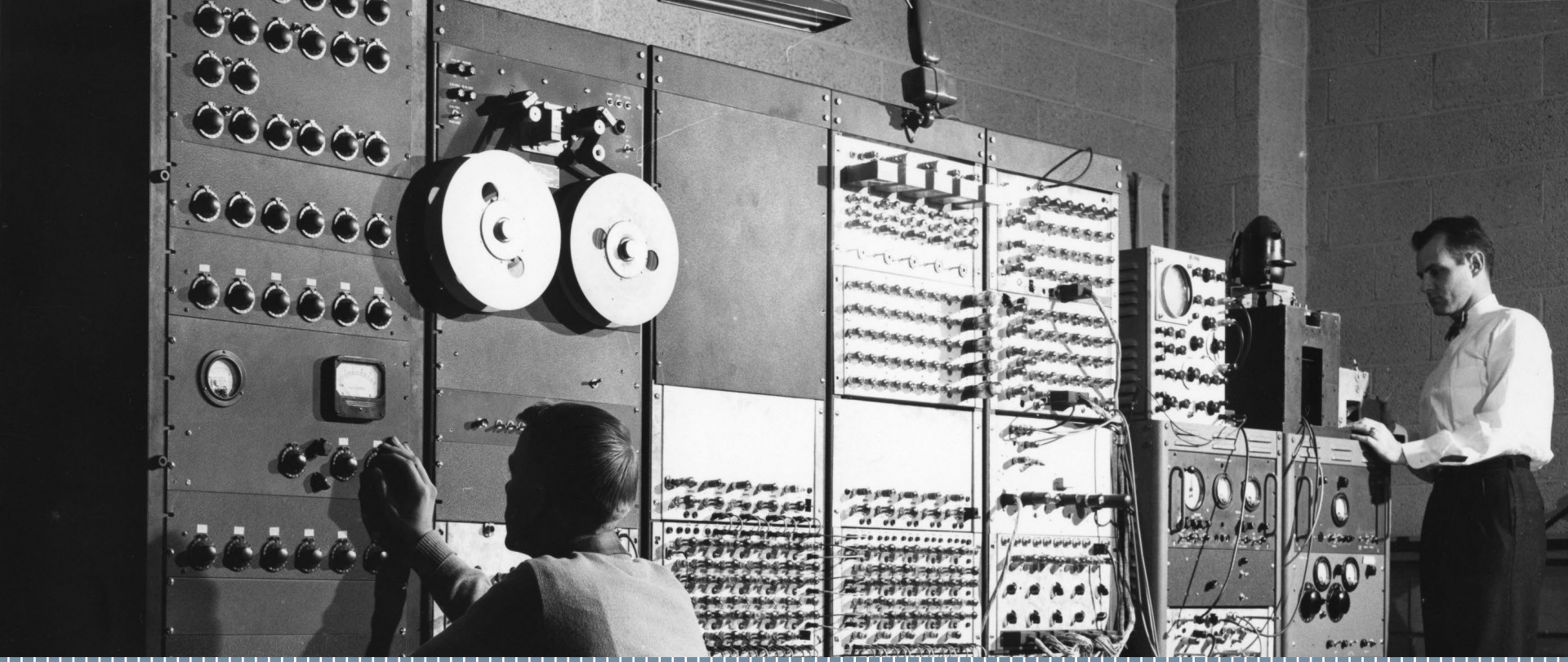

Prova finale API

8 giugno 2020 - Incontro di tutorato

Lorenzo Farinelli – lorenzo.farinelli@mail.polimi.it
Rodrigo Principe – rodrigo.principe@mail.polimi.it

Outline

- Contatti
- Deadline
- Valutazione e scadenze
- Specifiche del progetto
    - Comandi
    - Assunzioni
- Strumenti di sviluppo
    - GCC
    - GDB
    - Altri tool debugging
- Messaggi di errore verificatore
- Demo tool debugging
- Demo verificatore

Contatti

Domande burocratiche

Contattare il prof. Pradella – matteo.pradella@polimi.it

Domande di carattere generale

Se pensate sia di interesse generale: aula virtuale di Teams

Altrimenti contattare via mail il tutor di riferimento:

- Cognomi da E a LA (incluso): lorenzo.farinelli@mail.polimi.it
- Cognomi da LA (escluso) a P: rodrigo.principe@mail.polimi.it

Domande specifiche sull'implementazione

Contattare via mail il tutor di riferimento, per fissare un eventuale appuntamento

Deadline

Laureandi

- Per i laureandi di luglio la scadenza è alle ore **23:59** di **giovedì 9 luglio**

Non Laureandi

- Per tutti gli altri la scadenza è alle ore **23:59** di **giovedì 10 settembre**

Valutazione

- Sono previsti 6 task, composti ciascuno di:
  - 1 test pubblico : input e output sono disponibili
  - 3 test privati di difficoltà crescente

- Per superare l'esame è necessario passare almeno il test pubblico di ciascun task e il test privato più semplice

- Per ottenere la lode è necessario superare un task apposito

- Il file sottomesso per la valutazione deve essere lo stesso per tutti i task

- Se non si supera la prova o se si rifiuta la valutazione, è possibile ritentarla solamente nel successivo anno accademico

Annullamento della prova

La prova può essere annullata se identificati casi di:

- **plagio**, anche parziale, da altri progetti o da internet
- **distribuzione** del proprio elaborato
- **manomissione** della piattaforma di valutazione

- Non pubblicare il codice su repository pubblici

Progetto - edU

edU = **ed**itor di testo con **U**ndo multiple su terminale

ind – numero intero che indica l'indice di riga sulla quale vado ad operare

Comandi del editor:

- Change `(ind1,ind2)c`
- Delete `(ind1,ind2)d`
- Print `(ind1,ind2)p`
- Undo `(numero)u`
- Redo `(numero)r`
- Quit `q`

edU – No input validation

L'editor riceverà solo comandi corretti (**non ho bisogno della verifica input**)

Escluse a priori situazioni tipo:

- Range di indici non validi: **ind1 > ind2**
- Scrittura anteriore alla prima riga del file: **ind1 < 1**
- Indice non valido: **ind1** è l'indirizzo di una **riga non presente** nel testo

Attenzione in alcuni casi, tipo:

- delete con ind1 non presente nel testo è un commando valido, anche se non ha effetto

edU - Specifiche ed assunzioni

- `Undo` applicati in numero superiore ai cambiamenti fatti riportano il documento allo *stato iniziale*, cioè vuoto (undo in eccesso non hanno effetto)
- `Redo` applicati in numero superiore agli annullamenti fatti riportano il documento all'*ultima modifica* (redo in eccesso non hanno effetto)

- Gli input vengono forniti da standard input (`stdin`)
  La lunghezza delle stringhe non eccede i 1024 caratteri

- Non vanno considerate biforcazioni, se dopo un `undo` trovo un comando che modifica il testo, la cronologia delle modifiche viene azzerata

Strumenti di sviluppo

- Utilizzabile **qualsiasi** editor di testo o IDE (sconsigliati IDE complessi)
- I tools presentati sono sufficienti sia per l'implementazione
  sia per l'analisi e l'ottimizzazione del progetto
- Consigliato sistema operativo **Linux**, per emulare verificatore
- È possibile utilizzare solamente la **libreria standard** di C (no librerie esterne)

Strumenti di sviluppo – Macchina virtuale

- Macchina virtuale Linux
  https://polimi365-my.sharepoint.com/:u:/g/personal/10432943_polimi_it/EWdJHRrY7BtNnX1-WQIkQ-sBm2r8DJomIliWWVib6TYvRw?e=tTO5FY

- VirtualBox consigliato
  https://www.virtualbox.org/

- Attivare la Virtualization Technology (VT-x) da bios e VirtualBox
  https://youtu.be/3ZUVR22c978

Strumenti di sviluppo – GCC

Compilare il programma su terminale (senza opzioni)

```
gcc sorgente.c -o eseguibile
```

Per compilare il programma tramite gcc includendo le informazioni al debugging destinate a gdb (**opzione ggdb**):

```
gcc -ggdb sorgente.c -o eseguibile
```

Per simulare al meglio il verificatore usare anche le opzioni in **homepage** del **verificatore** sotto «**compilation commands**».

Strumenti di sviluppo – Pipeline e Redirezionamenti

Per testare la propria implementazione in locale sulla base dei **test pubblici**:

Fornire contenuto del file in input al programma

```
cat ./public_input_file | ./program
./programma < public_input_file
```

Scrivere output del programma su file

```
cat ./public_input_file | ./program > program_output
./programma < public_input_file > program_output
```

Confrontare contenuto di due file

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

(nessun output significa che i due file sono identici)

Strumenti di sviluppo – GDB

Strumento per eseguire debugging dell'eseguibile compilato.

Utile per ispezionare il valore di determinate variabili a run-time,
o monitorare il flusso di esecuzione

`gdb program_file`

- `run [ < input_file ] -` avvia esecuzione programma
- `list [ funzione / riga ] -` stampa codice vicino il break-point
- `break [ funzione / riga ]` - interrompi esecuzione nel punto specificato
- `print variable` – stampa valore della variabile specificata
- `continue` – continua esecuzione fino a prossimo break-point
- `next` – esegui prossima riga di codice, senza entrarne all'interno
- `step` – esegui prossima riga di codice, entrando nella funzione
- `bt / where / backtrace` – stampa stack del programma

Strumenti di sviluppo – Valgrind e Kcachegrind

- valgrind

```
valgrind eseguibile
```

https://medium.com/@sukhbeerdhillon305/using-valgrind-8c0f394339ac

- kcachegrind

```
valgrind --tool=callgrind eseguibile
kcachegrind callgrind.out.xxxx
```

https://developer.mantidproject.org/ProfilingWithValgrind.html

- Vademecum dei tools

https://docs.google.com/document/d/1d5yRqthRV49xNLGVBEgdWdw4_igCMaYcyB9izP4X6aQ/edit

Messaggi di errore

La piattaforma fornisce indicazioni su eventuali errori di esecuzione dei test:

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: implementazione non soddisfa i vincoli di tempo e memoria del test. Utilizzare valgrind per identificare le parti di codice più dispendiose o memory leaks; valutare strutture dati appropriate

Execution killed with signal 11: implementazione non soddisfa vincoli di memoria o si è verificato segmentation fault. Utilizzare valgrind in locale

Execution failed because the return code was nonzero: il main non ritorna 0 o il programma termina inaspettatamente. Debug locale

Demo kcachegrind

Demo verificatore online

Demo verificatore online

- Aprire il link del verificatore online https://dum-e.deib.polimi.it
- Inserire le proprie credenziali
- Aprire il task tutorial
- Scrivere sul proprio pc il codice C richiesto
- Verificare che funzioni **in locale** con file di **test pubblici**
- Caricare il codice ed eseguirlo
- Verificare output di esecuzione