# Prova Finale di Algoritmi e Strutture Dati

*A.A. 2018/2019*

*note generali*

# Introduzione

- Obiettivo: implementazione efficiente (e corretta!) di un algoritmo
- Logistica
  - codice sorgente sarà caricato su un server, compilato e fatto girare automaticamente
  - Scadenze (strette e vincolanti):
    - 10 luglio (ore 23.59) per i laureandi di luglio
    - 12 settembre (ore 23.59) per tutti gli altri
  - Tenete conto dei vincoli temporali per gestirvi!
    - Es. "inizio a pensarci a settembre" = fallimento quasi certo
- Esecuzione del progetto
  - implementazione nel linguaggio C
  - esclusivamente con libreria standard (libc)
  - no thread o tecniche di parallelizzazione

# Valutazione

- programma deve compilare e girare correttamente
  - verranno resi disponibili dei casi di test come prova per controllare il corretto funzionamento del programma
  - dopo il caricamento sul server, il programma viene fatto girare su test, divisi in 2 parti: pubblici e privati
- si misura la correttezza (risultati in uscita) e l'efficienza (tempi di risposta e memoria occupata) del programma su vari casi di test
- dipendentemente dai risultati sui casi di test, potrete **calcolare** il voto
  - il voto è assegnato in modo **automatico** in base a come vanno i test
- non c'è recupero, ma il numero di "appelli" (= sottomissioni) è praticamente illimitato

# Sito per la sottomissione del progetto

- https://dum-e.deib.polimi.it/
- Ogni studente avrà le proprie credenziali per accedere al sito

# Operazioni che si possono fare sul sito

- scaricare le specifiche del progetto
- caricare (*submit / sottoponi*) il file da compilare e lanciare
  - si può usare il nome del file che si vuole, viene rinominato in automatico
  - dopo il caricamento, in automatico il file viene compilato e testato su un insieme di test “pubblici”
- lanciare (*play / usa*) i test “privati”, quelli su cui viene valutato il progetto

# Gestione del codice

- Vi è **richiesto** di condividere il codice con il docente tramite GitHub (www.github.com)
- A questo fine occorre:
  - Registrarsi su github, usando l'email del Politecnico
    - l'email del Politecnico dà la possibilità di creare repository privati gratis
    - guardate questo link: https://education.github.com/pack
  - Creare un repository **privato**, chiamandolo "PFAPI19_<cognome>_<idnumber>"
  - "Invitare" il docente a condividere il repository
    - l'id del docente è matteo-g-rossi
- Se già non avete dimestichezza con git, online ci sono diversi tutorial
  - ma le operazioni che dovete fare voi per questo progetto sono molto semplici, di fatto solo delle "push" di codice sul repository

# Plagi

- progetto da svolgere singolarmente ed in **totale autonomia** (no a gruppi)
- siete responsabili del vostro codice, quindi vi consigliamo fortemente di:
  1. Non caricarlo in repository **pubblici**
  2. Non passarlo per "ispirazione" a colleghi
- controllo plagi automatizzato
- in caso di copiatura **tutti** i progetti coinvolti vengono **annullati**

# Scadenze

- Giovedì 12 settembre, ore 23.59
  - poi il sito per la sottomissione verrà chiuso

- Per i laureandi di luglio: mercoledì 10 luglio, ore 23.59
  - avvisate (mandando un email al docente) che volete avere la valutazione a luglio

# Calendario incontri tutoraggio

- Luogo: ufficio docente (ufficio 6, primo piano)
- giovedì 27/6, 14.00-17.00
- venerdì 5/7, 10.00-13.00
- venerdì 12/7, 14.00-17.00
- martedì 3/9, 10.00-13.00

# Un sistema di monitoraggio di relazioni tra elementi

Prova finale di Algoritmi e Strutture Dati AA 2018-19

# Il tema

- Si vuole implementare un meccanismo di monitoraggio di relazioni tra entità (per esempio persone) che cambiano nel tempo
- Si immagini, per esempio, un social network, in cui nuovi utenti possono registrarsi, e utenti esistenti possono cancellare il proprio account, diventare “amici” di altri utenti, rompere la relazione di amicizia, ecc.
- Le relazioni tra entità non sono necessariamente simmetriche. Per esempio, Alice può essere “amica” di Bruno, ma l’amicizia non è reciprocata (Bruno non è amico di Alice)

# Il tema (cont.)

- In maniera più astratta, il meccanismo monitora i seguenti fenomeni:
  - Una nuova entità  comincia ad essere monitorata
  - Una entità monitorata smette di esserlo
  - Una nuova relazione viene stabilita tra 2 entità monitorate
  - Una relazione esistente tra 2 entità monitorate cessa di esistere
- Ogni entità ha un nome identificativo (per esempio "Alice", "Bruno", "Carlo")
- Ci possono essere diversi tipi di relazioni tra entità, ognuna identificata da un nome (per esempio, "amico_di", "segue", "coetaneo_di")
- Ogni relazione ha un verso (per esempio, se Alice è "amico_di" Bruno, il verso della relazione è da Alice a Bruno, quindi Bruno è il “ricevente” della relazione), e non è necessariamente simmetrica
- A seguito di un apposito comando, il sistema restituisce, per ogni relazione, l’entità che “riceve” più relazioni (se ci sono più entità il cui numero di relazioni ricevute è massimo, queste vengono stampate in ordine crescente di identificativo)
- L’applicativo dovrà essere ottimizzato per gestire un grande numero di entità e istanze di relazioni, ma generalmente pochi tipi (identificativi) di relazione

# Il progetto

- Implementazione in linguaggio C standard (con la sola *libc*) di un programma che legge da *standard input* una sequenza di comandi, ognuno corrispondente ad un cambiamento nelle entità o nelle relazioni tra entità e, quando richiesto, produce su *standard output*, per ogni tipo di relazione monitorata, l'identificativo dell'entità che è il ricevente del maggior numero di istanze di quella relazione, e il numero di relazioni che l'entità riceve

# Comandi

- I comandi che possono essere letti sono i seguenti:
  - addent ⟨id_ent⟩
    - aggiunge un'entità identificata da "id_ent" all'insieme delle entità monitorate; se l'entità è già monitorata, non fa nulla
  - delent ⟨id_ent⟩
    - elimina l'entità identificata da "id_ent" dall'insieme delle entità monitorate; elimina tutte le relazioni di cui "id_ent" fa parte (sia come origine, che come destinazione)
  - addrel ⟨id_orig⟩ ⟨id_dest⟩ ⟨id_rel⟩
    - aggiunge una relazione – identificata da "id_rel" – tra le entità "id_orig" e "id_dest", in cui "id_dest" è il ricevente della relazione. Se la relazione tra "id_orig" e "id_dest" già esiste, o se almeno una delle entità non è monitorata, non fa nulla. Il monitoraggio del tipo di relazione "id_rel" inizia implicitamente con il primo comando "addrel" che la riguarda.
  - delrel ⟨id_orig⟩ ⟨id_dest⟩ ⟨id_rel⟩
    - elimina la relazione identificata da "id_rel" tra le entità "id_orig" e "id_dest" (laddove "id_dest" è il ricevente della relazione); se non c'è relazione "id_rel" tra "id_orig" e "id_dest" (con "id_dest" come ricevente), non fa nulla
  - report
    - emette in output l'elenco delle relazioni, riportando per ciascuna le entità con il maggior numero di relazioni entranti, come spiegato in seguito
  - end
    - termine della sequenza di comandi

# Osservazioni

- Gli identificativi (sia di entità che di relazione) sono sempre racchiusi tra ""
- Si assuma pure che ogni identificativo possa contenere solo lettere (maiuscole o minuscole), cifre, ed i simboli "_" e "-"
  - non serve controllare che gli identificativi ricevuti rispettino questa convenzione, la si può dare per scontata
- Tutti gli identificativi (sia delle entità che delle relazioni) sono "case sensitive", per cui "Alice" e "alice" sono identificativi diversi

# Osservazioni (cont.)

- L’output del comando *report* è una sequenza fatta 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 è *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
- Nessun comando ha output tranne *report*

# Esempio di sequenza di comandi di input, con corrispondente output

| input | output |
|---|---|
| addent "alice" | |
| addent "bruno" | |
| addent "carlo" | |
| addent "dario" | |
| report | none |
| addrel "carlo" "bruno" "amico_di" | |
| report | "amico_di" "bruno" 1; |
| addrel "carlo" "alice" "amico_di" | |
| report | "amico_di" "alice" "bruno" 1; |
| addrel "alice" "bruno" "amico_di" | |
| report | "amico_di" "bruno" 2; |

# Esempio di sequenza di comandi di input, con corrispondente output (cont.)

| input | output |
|---|---|
| addrel "bruno" "dario" "compagno_di" | |
| report | "amico_di" "bruno" 2; "compagno_di" "dario" 1; |
| delrel "carlo" "alice" "amico_di" | |
| report | "amico_di" "bruno" 2; "compagno_di" "dario" 1; |
| addrel "carlo" "alice" "compagno_di" | |
| report | "amico_di" "bruno" 2; "compagno_di" "alice" "dario" 1; |
| addrel "carlo" "bruno" "compagno_di" | |
| report | "amico_di" "bruno" 2; "compagno_di" "alice" "bruno" "dario" 1; |
| delent "alice" | |
| report | "amico_di" "bruno" 1; "compagno_di" "bruno" "dario" 1; |
| end | |

# Breve tutorial del sito

# Overview

- “Tutorial” è un semplicissimo problema, creato al solo scopo di sperimentare il funzionamento del sito
- Tutti gli altri sono i problemi da risolvere (i test che **devono** essere eseguiti con successo)

**Task overview**

| Task | Name | Time limit | Memory limit | Type | Files | Tokens |
|---|---|---|---|---|---|---|
| **Tutorial** | Tutorial | 1.000 second | 256 MiB | Batch | batch[.c] | Yes |
| **Monotone** | Monotone | 4.320 seconds | 200 MiB | Batch | Suite1[.c] | Yes |
| **DropOff** | DropOff | 6.963 seconds | 190 MiB | Batch | Suite2[.c] | Yes |
| **MixUp** | MixUp | 0.610 seconds | 6.00 MiB | Batch | Suite3[.c] | Yes |
| **Repeated** | Repeated | 1.139 seconds | 9.00 MiB | Batch | Suite4[.c] | Yes |
| **MultipleMixUp** | MultipleMixUp | 3.062 seconds | 20.0 MiB | Batch | Suite5[.c] | Yes |
| **MultipleRepeated** | MultipleRepeated | 5.340 seconds | 35.0 MiB | Batch | Suite6[.c] | Yes |
| **Lode** | Lode | 1.640 seconds | 23.0 MiB | Batch | Lode[.c] | Yes |

# “Tutorial”

- Problema: leggere da standard input 2 numeri interi separati da uno spazio; produrre su standard output la loro somma

## Some details

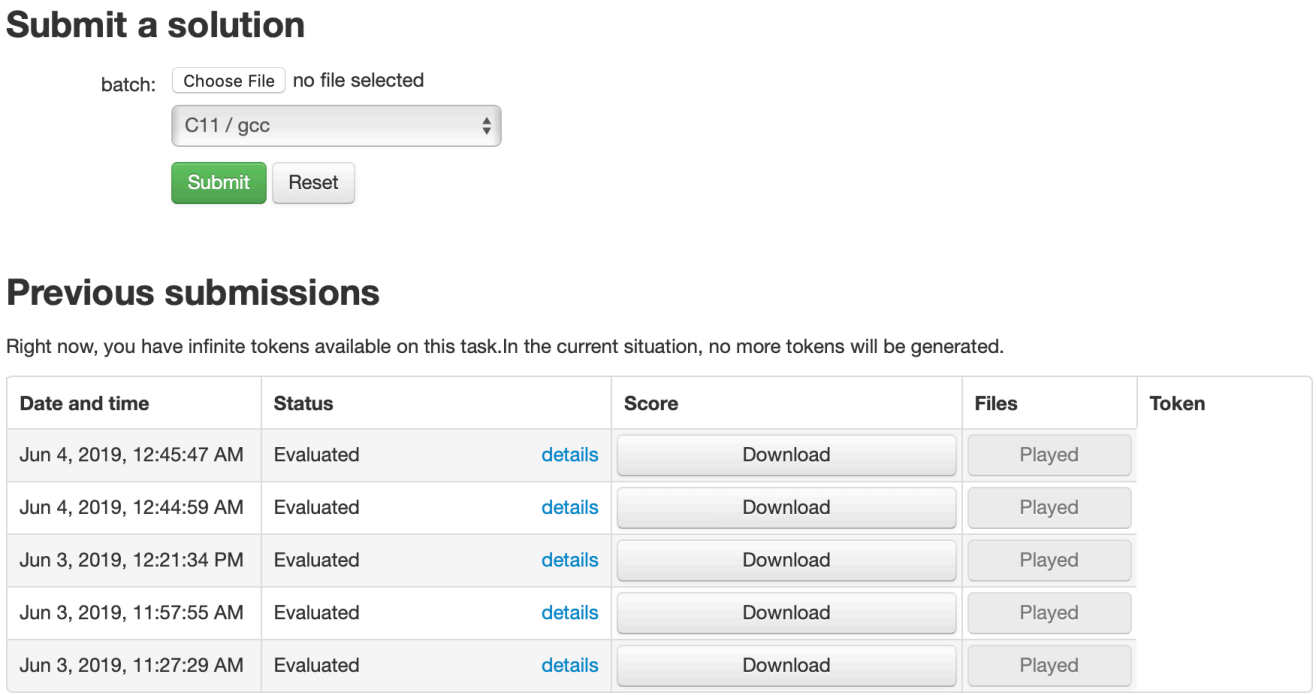
| | | |
|---|---|---|
| **Type** | Batch | |
| **Time limit** | 1 second | |
| **Memory limit** | 256 MiB | |
| **Compilation commands** | C11 / gcc | `/usr/bin/gcc -DEVAL -std=c11 -O2 -pipe -static -s -o Tutorial Tutorial.c -lm` |
| **Tokens** | You have an infinite number of tokens for this task. | |

# Passi da fare

- Step 1: creare il file Tutorial.c
- Step 2: compilare il file con la riga di comando (presa dal sito)
  /usr/bin/gcc -DEVAL -std=gnu11 -O2 -pipe -static -s -o Tutorial Tutorial.c -lm
- Step 3: preparate un file di test, per esempio Tutorial_test.in, con una riga di testo, in cui ci sono 2 interi separati da uno spazio; preparate anche un file Tutorial_test.out con il risultato atteso della computazione (una riga con il risultato della somma)
- Step 4: lanciate il comando
  cat Tutorial_test.in | ./Tutorial > Tutorial_test.res
  - crea un file, Tutorial_test.res, con il risultato della computazione

# Passi da fare (cont.)

- Step 5: confrontate il risultato atteso con quello ottenuto mediante il comando diff Tutorial_test.out Tutorial_test.res
  - se non ci sono differenze il risultato è la stringa vuota
  - il server usa il comando *wdiff* invece del comando *diff*
- Step 6: caricate ("sottoponi") il file Tutorial.c sul sito, e lanciate ("usa") i test



# Possibili strumenti utili

- valgrind (memory debugging, profiling)
  - http://valgrind.org
- gdb (debugging)
  - https://www.gnu.org/software/gdb/
- AddessSanitizer (ASan, memory error detector)
  - https://github.com/google/sanitizers/wiki/AddressSanitizer

# Calcolo del 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 il subtask privato più semplice di ogni task
    - 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