Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi – 11 Febbraio 2015

1. PER GLI STUDENTI DI SISTEMI OPERATIVI E RETI: è necessario sostenere e consegnare entrambi gli esercizi. Sarà attribuito un unico voto su tutta la prova.

2. PER GLI STUDENTI DI SISTEMI OPERATIVI: si può sostenere solo uno dei due esercizi se si è già superata in un appello precedente la corrispondente prova. Il tempo a disposizione in questo caso è di 2 ORE.

Troverete sul vostro Desktop una cartella chiamata "CognomeNomeMatricola" che contiene la traccia dell'elaborato ed eventuali altri file utili per lo svolgimento della prova. Ai fini del superamento della prova è indispensabile rinominare tale cartella sostituendo "Cognome" "Nome" e "Matricola" con i vostri dati personali. Ad esempio, uno studente che si chiama Alex Britti ed ha matricola 66052 dovrà rinominare la cartella "CognomeNomeMatricola" in "BrittiAlex66052".

Non saranno presi in considerazione file non chiaramente riconducibili al proprio autore. E' possibile caricare qualsiasi tipo di materiale didattico sul desktop nei primi 5 minuti della prova.

Si consiglia di salvare SPESSO il proprio lavoro.

ESERCIZIO 1 (Linguaggi di scripting)

Il file ***campionato*** contiene informazioni relative ai risultati delle partite giocate in diverse giornate di campionato. In particolare, il file è così strutturato:

campionato

```
GIORNATA DI CAMPIONATO 1

Sassuolo-Udinese 1-1
Torino-Milan 1-1
Inter-Genoa 3-1
Atalanta-Chievo 1-1
Cagliari-Cesena 2-1
Fiorentina-Palermo 4-3
Verona-Parma 3-1
Roma-Lazio 2-2
Sampdoria-Empoli 1-0
Napoli-Juventus 1-3

GIORNATA DI CAMPIONATO 2

Empoli-Inter 0-0
Palermo-Roma 1-1
lazion-Napoli 0-1
Cesena-Torino 2-3
Chievo-Fiorentina 1-2
Genoa-Sassuolo 3-3
Milan-Atalanta 0-1
Parma-Sampdoria 0-2
udinese-Cagliari 2-2
Juventus-Verona 4-0

GIORNATA DI CAMPIONATO 3

Cagliari-Sassuolo 2-1
Lazio-Milan 3-1
Verona-Atalanta 1-0
Inter-Torino 0-1
Juventus-Chievo 2-0
Parma-Cesena 1-2
Sampdoria-Palermo 1-1
Fiorentina-Roma 1-1
Empoli-Udinese 1-2
Napoli-Genoa 2-1
```

Ad esempio, la prima riga del file

```
Sassuolo-Udinese 1-1
```

indica che la partita `Sassuolo-Udinese` è stata pareggiata 1-1.
Si scriva uno script perl capace di conteggiare i punteggi complessivi ottenuti da tutte le squadre che prendono parte al campionato. In particolare, il punteggio di ciascuna squadra è così calcolato:

- ad una partita vinta sono attribuiti 3 punti
- ad una partita pareggiata è attribuito 1 punto

Lo script per deve:

1. leggere **da linea di comando** il nome del file contenente i risultati delle partite del campionato (ad esempio, il file ***campionato***),
2. **calcolare i punteggi** per tutte le squadre che partecipano al campionato,
3. **stampare** su un nuovo file ***risultati*** la classifica delle squadre, ordinata in senso decrescente rispetto al punteggio ottenuto. Quindi, il file ***risultati*** dovrà contenere una lista di risultati del tipo *nome_squadra:punteggio_totale* ordinata rispetto al punteggio in classifica (a partire dalla squadra che ha totalizzato un punteggio più alto, per finire con la squadra che ha ottenuto il punteggio più basso in classifica).

ESERCIZIO 2 (Programmazione multithread)
Si deve progettare una struttura dati per la gestione concorrente di una classifica di campionato e dei corrispondenti risultati calcistici.

A tal proposito, una `Squadra` è data dalla struttura dati:

```
class Squadra
{
        …
    String nome;
        …   ← (puntini che suggeriscono che puoi estendere la struttura per come ritieni opportuno)
}
```

Mentre una `Partita` consiste in:

```
class Partita
{
        …
        Squadra diCasa;
        Squadra ospite;
        int goalDiCasa;
        int goalOspite;
        int nGiornata;
        boolean risultatoDefinitivo;
        …   ← (puntini che suggeriscono che puoi estendere la struttura per come ritieni opportuno)
}
```

Un `RegistroCampionato` modella invece un campionato a girone tra N squadre, condotto in più giornate di gioco.
Un `RegistroCampionato` fornisce tutte le funzionalità necessarie per: l'aggiunta di risultati di singole partite; la consultazione di risultati; la consultazione della classifica ufficiale o provvisoria.
La progettazione di dettaglio di `RegistroCampionato` è a carico dello studente. Il `RegistroCampionato` può contenere anche informazioni su partite che sono ancora in svolgimento (per cui il corrispondente risultato non è definitivo).

Le operazioni che si chiede di implementare su un `RegistroCampionato` devono essere thread-safe. Ad esempio, durante una giornata di campionato, mentre sono in corso le differenti partite, deve essere possibile aggiornare e consultare i risultati parziali di ciascuna partita in maniera concorrente ed evitando inconsistenze (si immagini ad esempio una situazione in cui più cronisti sono assegnati a rispettive partite e si occupano di aggiornare i risultati del rispettivo evento sportivo mano a mano che questo evolve).
I metodi pubblici che `RegistroCampionato` deve implementare sono:

`void aggiungiPartita(Partita p)`
Aggiunge la partita `p` a tutte quelle già registrate nel `RegistroCampionato`. Una nuova partita comincia dal risultato di 0 a 0 e viene inizialmente considerata *in corso* (`risultatoDefinitivo = false`).

`void registraGol(Partita p, boolean diCasaOppureOspite)`
Assumendo che `p` sia già presente nel `RegistroCampionato`, si deve aggiornare il risultato di `p` attribuendo un gol in più alla squadra di casa ovvero alla squadra ospite in accordo al booleano `diCasaOppureOspite`.

`void terminaPartita(Partita p)`
Pone `p.risultatoDefinitivo = true`. Non deve essere possibile aggiornare i gol segnati in una partita se questa è terminata.

`Squadra[] classificaUfficiale()`. Restituisce un array di squadre ordinate in base alla classifica ufficiale del campionato, considerando solo i risultati delle partite delle giornate per le quali **tutte le partite risultano terminate**. Ad esempio, se per la decima giornata risultano esserci ancora 3 partite in corso su 10, non bisogna considerare, nel calcolo del punteggio, nessuna partita di questa giornata, anche se qualcuna di esse risulta essere già terminata.

`Squadra[] classificaProvvisoria()`. Restituisce un array di squadre ordinate in base alla classifica attuale del campionato, considerando i risultati di tutte le partite che risultano registrate anche se facenti parte di una giornata di gioco ancora in corso.

Calcolo della classifica: le squadre vengono ordinate in base al proprio punteggio in classifica. Il punteggio di una squadra è calcolato in base alle partite vinte e pareggiate. Una partita è considerata vinta se si sono segnati più gol di quelli segnati dalla squadra avversaria. Per le partite vinte vengono attribuiti 3 Punti, mentre per le partite pareggiate viene attribuito 1 punto.

***E' parte integrante** di questo esercizio completare le specifiche date nei punti non esplicitamente definiti, introducendo o estendendo tutte le strutture dati che si ritengano necessarie, e risolvendo eventuali ambiguità. Non è consentito modificare il prototipo dei metodi se questo è stato fornito.*
***Si può svolgere questo esercizio in un qualsiasi linguaggio di programmazione a scelta** dotato di costrutti di supporto alla programmazione multi-threaded (esempio, C++ con libreria JTC, Java).*
E' consentito usare qualsiasi funzione di libreria di Java 6.
Non è esplicitamente richiesto di scrivere un main() o di implementare esplicitamente dei thread di prova, anche se lo si suggerisce per testare il proprio codice prima della consegna.