Esercitazione di laboratorio n. 9

(Caricamento sul portale entro le 23.59 del 12/12/2016 dell'esercizio 1 e di almeno uno tra gli esercizi 2 e 3)

Esercizio n.1: Corpo Libero

Il corpo libero è una specialità della ginnastica artistica/acrobatica in cui l'atleta deve eseguire una sequenza di esercizi, ognuno caratterizzato da un corrispettivo punteggio atteso e relativo grado di difficoltà.

Ai fini dell'esercizio corrente, si considerino le seguenti indicazioni per la costruzione di un programma di corpo libero:

- gli esercizi sono divisi in categorie
- per ogni categoria c_i occorre eseguire almeno min_i elementi
- per ogni categoria c_i non possono essere eseguiti più di max_i elementi
- ogni ripetizione di un esercizio, indipendentemente dalla categoria, ne riduce il punteggio della metà del valore iniziale. Dalla terza esecuzione il punteggio vale quindi 0
- la difficoltà complessiva del programma non deve superare un valore (intero) `D`
- il punteggio atteso complessivo deve essere il più alto possibile.

L'elenco degli elementi tra cui è possibile scegliere è riportato in un file di testo (`elementi.txt`), il cui formato è il seguente:

- sulla prima riga è riportato il numero di categorie `K`
- seguono `K` sezioni, una per categoria, organizzate come segue:
  - sulla prima riga è riportata la quaterna `<nomeCat><numElem><min><max>`
  - seguono `numElem` righe riportanti i dettagli di ogni elemento
  - ogni elemento è descritto dalla terna `<nomeElem><punti><difficoltà>`

Ai fini di migliorare la leggibilità del file d'esempio, le varie sezioni del file sono evidenziate sfruttando una indentazione variabile.

Scrivere un programma in C che, dato un file il file di elementi e il valore `D`, generi un possibile programma di corpo libero che rispetti tutti i vincoli di cui sopra.

È richiesto di risolvere il problema in (almeno) due varianti:

- un algoritmo in grado di individuare una soluzione ottima, sulla base dei valori acquisiti
- uno, o più, approcci *greedy* in grado di trovare una soluzione ammissibile, ma non necessariamente ottimale, sulla base dei valori acquisiti.

Esercizio n.2: Ranking

Si supponga di dover gestire la classifica di un torneo in cui valgono le seguenti regole:

- ogni partecipante entra nel torneo con un numero di punti iniziali pari a 10
- un partecipante può essere aggiunto al torneo in qualsiasi momento
- un partecipante può essere eliminato dal torneo in qualsiasi momento
- il torneo evolve facendo sfidare i due partecipanti col minor numero di punti:
  - chi "vince" una sfida, ottiene il 25% dei punti dell'avversario
  - chi "perde" una sfida, perde il 25% dei propri punti
  - chi esaurisce i punti è eliminato dal torneo

Si realizzi un programma C che, attraverso un'apposita interfaccia utente, permetta di gestire l'evoluzione della classifica del torneo, nelle condizioni descritte sopra, appoggiandosi ad una struttura dati di tipo coda a priorità (`PQ`). Il tipo `Item` contenuto nella coda sia progettato per memorizzare le informazioni relative a ogni partecipante ritenute opportune.

Le operazione permesse devono essere quelle di:

- stampa dello stato della classifica
- inserimento di un nuovo partecipante
- eliminazione di un partecipante
- evoluzione della classifica (una singola sfida, il cui esito è generato casualmente)
- caricamento di dati da file (in una coda inizialmente vuota)
- salvataggio dei dati su file.

In questo esercizio, il programma deve essere realizzato su tre moduli distinti:

- l'interfaccia utente (il client)
- un modulo `PQ` con le funzioni per la gestione della coda
- un modulo `Item` con le funzioni per la gestione dei singoli dati.

Per quanto riguarda il modulo `PQ`, lo si realizzi come "quasi ADT" (basato su variabili globali), mediante lista concatenata non ordinata.

Nota:
Per la generazione di un numero casuale si consiglia di utilizzare la funzione `rand()`, che ritorna un numero intero casuale compreso tra 0 e `RAND_MAX`. Una risposta casuale `R` (in sostanza equiprobabile) tra 0 e 1 potrebbe essere generata con:

```
R = rand()  < RAND_MAX/2;
```

La funzione `rand()` genera in realtà un numero pseudo-casuale, in quanto, ripetendo il programma, si ripetono generano gli stessi numeri dell'esecuzione precedente (utilissimo, peraltro, per fare debug del programma).
Per evitare questo, occorre inizializzare il generatore di numeri casuali, ad ogni esecuzione, con un "seme" diverso dalla precedente esecuzione.
A tale scopo, includere la libreria `time.h` e aggiungere la seguente riga di codice in apertura del main:

```
srand((unsigned int)time(NULL));
```

Esercizio n.3: Ricettario (rivisto ulteriormente)
Si consideri la medesima situazione introdotta nell'esercizio 3 del laboratorio 7 e ripresa nell'esercizio 2 del laboratorio 8.
Sulla base delle specifiche riportate nel testo del laboratorio 08, si imposti il medesimo problema realizzando sia il modulo `ingredienti` sia il modulo `ricette` come ADT di prima classe (si utilizzino per le collezioni, a scelta, vettori o liste concatenate ordinati o non ordinati).