Esercitazione di laboratorio n. 7

Esercizio n. 1: Gioco di ruolo (multi-file, con ADT)
[Esercizio guidato: si forniscono architettura dei moduli ed esempi di file .h]

A partire dal codice prodotto per l'esercizio n. 3 del laboratorio 6, lo si adatti così che sia il modulo `personaggi` sia il modulo `inventario` risultino ADT

La specifica "nessuna statistica può risultare minore di 1, indipendentemente dagli eventuali *malus* cumulativi dovuti alla scelta di equipaggiamento" può essere interpretata nelle seguenti 2 modalità, entrambe accettate:

- si impedisce la scelta di un equipaggiamento se il malus porta almeno una delle statistiche sotto il valore 1
- si ammette la scelta di malus che portano almeno una statistica a valori negativi o nulli, ma in questo caso questa statistica viene "mascherata" in fase di stampa, dove si stampa il valore fittizio 1, quando il valore è negativo o nullo.

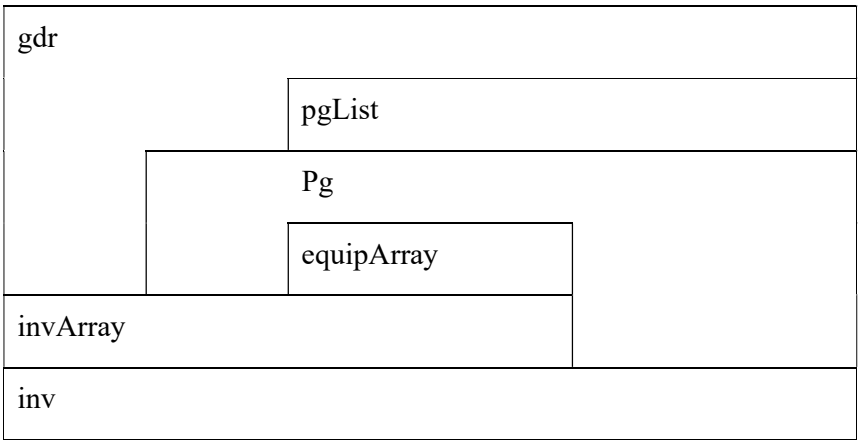
Si tenga conto che esistono:
- un tipo di dato per un oggetto e un tipo di dato per un vettore di oggetti (inventario)
- un tipo di dato per un personaggio e un tipo di dato per una lista di personaggi
- un tipo di dato per un vettore di (riferimenti a) oggetti (equipaggiamento)

Si chiede di realizzare tutti i tipi di dato come ADT, utilizzando:
- la versione "quasi ADT" (`struct` visibile) per i tipi personaggio (`pg_t`) e oggetto (`inv_t`)
- la versione "ADT di I classe" per le collezioni, cioè il vettore di oggetti (`invArray_t`), la lista di personaggi (`pgList_t`) e gli equipaggiamenti (`equipArray_t`).

Per i riferimenti ad oggetti si evitino i puntatori, in quanto richiederebbero accesso a una struttura dati interna (all'ADT vettore di oggetti). Si utilizzino invece gli indici: ad un dato oggetto si fa quindi riferimento mediante un intero (progressivo a partire da 0) che ne identifica la posizione nel vettore dell'inventario.

Pur potendo unificare l'ADT oggetto e il vettore inventario in un unico modulo, come pure l'ADT personaggio e l'ADT lista di personaggi, si consiglia di realizzare un modulo per ognuno degli ADT. Si realizzeranno quindi 5 file `.c` (`pg.c`, `inv.c`, `pgList.c`, `invArray.c`, `equipArray.c`) e 5 corrispettivi file `.h` per i moduli, più un `.c` per il client (ad esempio `gdr.c`).
I moduli `pg` ed `inv` realizzano tipi di dato composti per valore (sono quindi simili al tipo/ADT `Item` spesso usato in altri problemi): saranno ovviamente di dimensione ridotta.
I moduli per collezioni di dati dovranno fornire operazioni di creazione/distruzione, più eventuali operazioni di input o output, ricerca, modifica e/o cancellazione. Eventuali altre operazioni possono essere fornite qualora ritenute necessarie.

Architettura proposta:
La soluzione all'esercizio va realizzata secondo l'architettura dei moduli rappresentata nella seguente figura:



Lo schema mostra (dal basso in alto):

- gli elementi dell'inventario (modulo `inv`), un item quasi ADT, tipo `inv_t`, composto da 3 campi (di cui uno, un secondo tipo `struct`, `stat_t`, anch'esso quasi ADT) contiene le statistiche da leggere e aggiornare. Si consiglia di vedere `inv.c` come esempio di quasi ADT Item (non vuoto!) contenente operazioni elementari sui tipi di dato gestiti
- il vettore di elementi (`invArray`), realizzato come ADT vettore dinamico di elementi `inv_t`.
- il vettore di riferimenti (mediante indici) a elementi dell'inventario (`equipArray`) viene realizzato come semplice `struct` dinamica, contenente un vettore di interi di dimensione fissa (non allocato dinamicamente). Il modulo dipende da (è client di) `invArray`, solamente in quanto i riferimenti mediante indici sono relativi a un ADT di tale modulo
- il quasi ADT (un item) `pg_t` (modulo `pg`). Il modulo è client di `inv`, `invArray` e `equipArray`. Si noti che il quasi ADT contiene un campo ADT (`equip`) riferimento a un `equipArray_t`. Si tratta quindi di un quasi ADT di tipo 4 (in tal senso rappresenta un'eccezione rispetto alla prassi consigliata, non evitabile in base alle specifiche dell'esercizio), avente cioè un campo soggetto ad allocazione e deallocazione. L'allocazione (`equipArray_init`) viene gestita nella funzione di lettura da file (`pg_read`), mentre per la deallocazione si è predisposta la funzione `pg_clean`, che chiama semplicemente la `equipArray_free` (il tipo `pg_t` non va deallocato, in quanto quasi ADT)
- l'ADT `pgList_t` (modulo `pgList`) è un ulteriore ADT di prima classe, che realizza una lista di elemento del modulo `pg`
- il modulo principale (`gdr`) è client di `pgList`, `pg` e di `invArray`.

Si allegano i `.h` dei vari moduli e il `.c` di `gdr`. A scelta, si possono eventualmente modificare i nomi di tipi e funzioni, nonché definizioni di tipi e parametri delle funzioni (pur di rispettare l'architettura proposta e le richieste).

Questo non è l'unico schema realizzabile, ma deriva dall'aver effettuato certe scelte di modularità e di ripartizione delle operazioni tra moduli. Si consiglia di esaminare le dipendenze tra i moduli, nonché le scelte fatte nell'assegnare operazioni e funzioni ai singoli moduli.
SI noti che le funzioni che gestiscono i quasi ADT in alcuni casi ricevono e/o ritornano `struct`, in altri casi riferimenti (puntatori) a `struct` (ad esempio le funzioni di input/output da file sono state spesso predisposte in modo da ricevere puntatori alla `struct` che riceve o da cui si prendono i dati coinvolti nell'IO. Sono possibili altre scelte (es. le `struct` passate sempre per valore).

ATTENZIONE
Si ricorda che un ADT di I classe NASCONDE i dettagli interni di un dato: NON E' QUINDI POSSIBILE A UN CLIENT ACCEDERE A TALI DETTAGLI: *non sarà possibile, quindi la modifica dell'equipaggiamento di un dato personaggio da parte di un client in modo diretto, cioè ottenendo il puntatore all'elemento in lista, per accedere ai campi da modificare.* Il client dovrà quindi, ad esempio, chiamare una opportuna funzione (fornita dal modulo `pgList`) avente come parametri il codice di un personaggio. Tale funzione, all'interno, effettuerà una ricerca dell'atleta e ne modificherà l'esercizio selezionato NON in modo diretto, ma chiamando, ad esempio, una funzione (fornita dal modulo `equipArray`), che riceve come parametri il nome dell'oggetto di interesse e il tipo di operazione da eseguire. Quest'ultima funzione cerca l'oggetto e modifica lo stato della struttura dati di conseguenza.

Esercizio n.2: Corpo libero

Il corpo libero è una specialità della ginnastica artistica/acrobatica in cui l'atleta deve eseguire una sequenza di elementi senza l'ausilio di attrezzi, ad eccezione della pedana di gara.
In un programma di corpo libero il ginnasta è tenuto ad eseguire una serie di passaggi acrobatici, detti diagonali.
Ogni diagonale è composta da uno o più elementi.
Ogni elemento è descritto da una serie di parametri:

- nome dell'elemento (una stringa di massimo 100 caratteri senza spazi)
- tipologia: l'elemento può essere un elemento acrobatico avanti [2], acrobatico indietro [1] o di transizione [0]
- direzione di ingresso: il ginnasta può entrare in un elemento frontalmente [1] o di spalle [0]
- direzione di uscita: il ginnasta può uscire da un elemento frontalmente [1] o di spalle [0]
- requisito di precedenza: l'elemento può essere eseguito come primo di una sequenza [0] o deve essere preceduto da almeno un altro elemento [1]
- finale: l'elemento non può essere seguito da altri elementi [1] o meno [0]
- valore: il punteggio ottenuto dall'atleta per la corretta esecuzione di un elemento (reale)
- difficoltà: la difficoltà di esecuzione dell'elemento (intero).

Gli elementi sono memorizzati in un file di testo (`elementi.txt`) in ragione di uno per riga. Il numero di elementi è riportato sulla prima riga del file.

Perché due elementi possano essere eseguiti in sequenza, la direzione di uscita del primo elemento deve coincidere con la direzione di ingresso del secondo elemento. Un ginnasta inizia una diagonale sempre frontalmente. La difficoltà di una diagonale è definita come la somma delle difficoltà degli elementi che la compongono. La difficoltà del programma di gara è data dalla somma delle difficoltà delle diagonali che lo compongono.

Ai fini dell'esercizio, si considerino le seguenti regole:

- il ginnasta deve presentare 3 diagonali
- il ginnasta deve includere almeno un elemento acrobatico in ogni diagonale
- il ginnasta deve includere almeno un elemento acrobatico avanti e almeno un elemento acrobatico indietro nel corso del suo programma, ma non necessariamente nella stessa diagonale
- il ginnasta deve presentare almeno una diagonale in cui compaiono almeno due elementi acrobatici in sequenza
- se il ginnasta include un elemento finale di difficoltà 8 o superiore nell'ultima diagonale presentata in gara, il punteggio complessivo della diagonale viene moltiplicato per 1.5
- ogni diagonale può contenere al massimo 5 elementi
- ogni diagonale non può avere difficoltà superiore a un dato valore DD
- il programma complessivamente non può avere difficoltà superiore a un dato valore DP.

Si scriva un programma in C in grado di identificare la sequenza di diagonali che permettono al ginnasta di ottenere il punteggio più alto possibile, dato un set di elementi disponibili e il valore dei due parametri DD e DP.

Valutazione: entrambi gli esercizi saranno oggetto di valutazione
Scadenza caricamento di quanto valutato: entro le 23:59 del 10/01/2026.