**Esercizio n. 2: Collane e pietre preziose (versione 3)**

Si risolva l'esercizio n.1 del laboratorio 8 mediante il paradigma della memoization. Si richiede soltanto di calcolare la lunghezza massima della collana compatibile con le gemme a disposizione e con le regole di composizione.

Suggerimento: affrontare il problema con la tecnica del divide et impera, osservando che una collana di lunghezza P può essere definita ricorsivamente come:

- la collana vuota, formata da P=0 gemme
- una gemma seguita da una collana di P-1 gemme.

Poiché vi sono 4 tipi di gemme Z, R, T, S, si scrivano 4 funzioni `fZ`, `fR`, `fT`, `fS` che calcolino la lunghezza della collana più lunga iniziante rispettivamente con uno zaffiro, un rubino, un topazio e uno smeraldo avendo a disposizione `z` zaffiri, `r` rubini, `t` topazi e `s` smeraldi. Note le regole di composizione delle collane, è possibile esprimere ricorsivamente il valore di una certa funzione `fX` sulla base dei valori delle altre funzioni.
Si presti attenzione a definire adeguatamente i casi terminali di tali funzioni, onde evitare di ricorre in porzioni non ammissibili dello spazio degli stati.
Si ricorda che il paradigma della memoization prevede di memorizzare le soluzioni dei sottoproblemi già risolti in opportune strutture dati da progettare e dimensionare e di riusare dette soluzioni qualora si incontrino di nuovo gli stessi sottoproblemi, limitando l'uso della ricorsione alla soluzione dei sottoproblemi non ancora risolti.

**Esercizio n. 3: 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 8, 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.

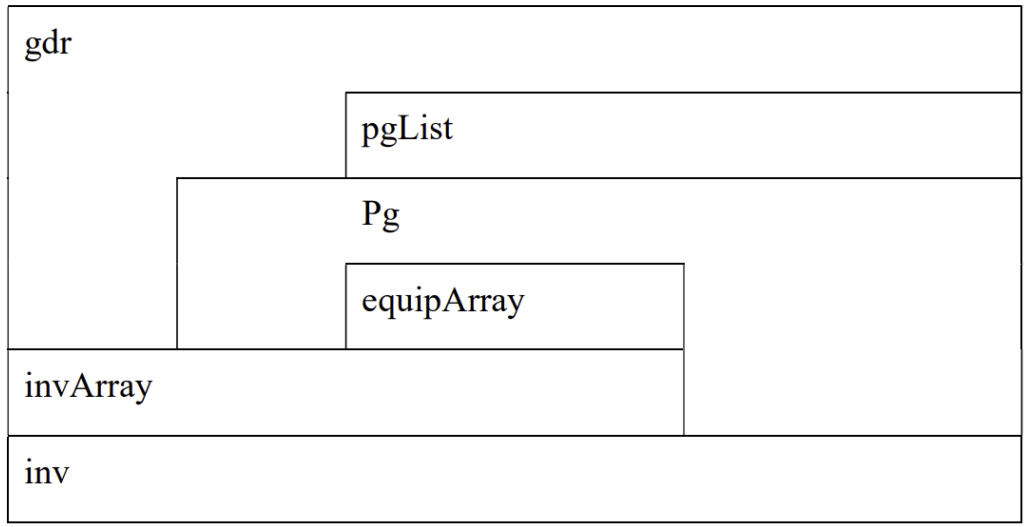
Si tenga conto che esistono:
- un tipo di dato per un oggetto e un tipo di dato per un vettore di oggetti (inventario)
- un tipo di dato per un personaggio e un tipo di dato per una lista di personaggi
- un tipo di dato per un vettore di (riferimenti a) oggetti (equipaggiamento)

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.

***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).