Progetto Esame Programmazione I

Canali per le domande: ricevimento, email: francesco.santini@dmi.unipg.it, Telegram: @safran

5 Dicembre 2016

Si realizzi un programma in linguaggio C che consiste dei seguenti tre file:

- `progexam.c` contiene solo la definizione della funzione *main()*.
- `mylib.c` contiene le definizioni delle funzioni che implementano il gioco.
- `mylib.h` contiene le dichiarazioni delle funzioni definite in `mylib.c` (solo quelle non *static*) e le definizioni del tipo delle strutture dati utilizzate.

La storia. In un mondo parallelo e fantastico, il Mago Oberon "Calzino Bucato" deve attraversare a piedi un percorso denso di insidie per poter arrivare finalmente a casa, nella sua torre magica, dove rilassarsi con un bel bicchiere di ippocrasso. In questo progetto dovrete realizzare un piccolo gioco dove, nella prima fase si costruisce il percorso, mentre nella seconda fase Oberon lo attraversa affrontando le insidie generate precedentemente, durante il processo di creazione. Oberon si difende con la spada e con i suoi incantesimi.

progexam.c. Questo file contiene solo la funzione *main()*, il cui compito è stampare un menu di scelte verso il giocatore ed aspettare il suo comando. Le possibili scelte sono: *1* crea percorso, *2* muovi Oberon, *3* termina gioco. *Suggerimento*: utilizzare un ciclo *do...while* per stampare il menu (dato che deve essere stampato per lo meno una volta), leggere la scelta del giocatore da tastiera (*scanf()*), e con uno *switch* eseguire i comandi per effettuare la scelta richiesta. In caso il comando non sia 1-2-3, stampare un messaggio al giocatore che il comando è sbagliato, e poi ristampare il menu. Nel caso la scelta sia 1 o 2, chiamare la corrispondente funzione definita in *mylib.c*, cioè *crea_percorso()*, *muovi_Oberon()*, e *termina_gioco()*.

mylib.h. Questo file deve contenere solamente le dichiarazioni delle funzioni definite in mylib.c come non statiche (vedere paragrafo successivo), e le definizioni dei tipi necessari alla libreria, cioè la *struct Oberon* e la *struct Terra*. La *struct Oberon* contiene *i) short borsa_oro*, *ii) short punti_ferita*, *iii) short incantesimi*, e *iv) short pozione_guaritrice*.

La *struct Terra* contiene per lo meno i campi *i) enum Tipo_terra tipo_terra*, *ii) enum Tipo_mostro tipo_mostro*, *iii) short tesoro*, *iv) struct Terra* terra_successiva*.

Il tipo *enum Tipo_terra* deve avere per lo meno i possibili valori *deserto, foresta, palude, villaggio, pianura* (anche questo tipo *enum* va definito dentro *mylib.h*).

Il tipo *enum Tipo_mostro* deve avere per lo meno i possibili valori *nessuno, scheletro, lupo, orco, drago* (anche questo tipo *enum* va definito dentro *mylib.h*).

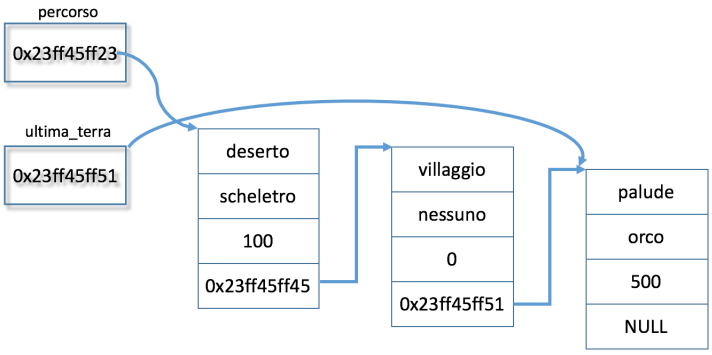

Figura 1: Esempio di lista dinamica con 3 terre.

mylib.c. Questo file rappresenta il nucleo del progetto: esso contiene tre variabili globali, una *static struct Terra* percorso* (inizializzata a *NULL*) che punta alla prima terra del percorso, una *static struct Terra* ultima_terra* (inizializzata a *NULL*) che punta all' ultima terra del percorso, e una variabile *static struct Oberon oberon* che rappresenta *Oberon*. Le variabili devono essere *static* perché modificabili solamente tramite le funzioni della libreria (solo all'interno di *mylib.c*).

La lista delle terre che rappresentano il percorso è un lista dinamica, dato che a priori non sappiamo il numero di terre nel percorso (è il giocatore a sceglierne il numero passo dopo passo): il puntatore *percorso* punta alla prima terra, ed ogni terra che viene aggiunta attraverso la funzione *ins_terra()* (vedere dopo), richiamata dalla *crea_percorso()*, viene allocata in memoria dinamica utilizzando *malloc()*. Quando si deve eliminare dalla memoria dinamica, utilizzare *free()* sul puntatore della terra. All'interno di ogni terra, il campo *struct Terra* terra_successiva* punta alla terra successiva nel percorso (deve essere *NULL* in caso sia l'ultima terra del percorso). Alla fine di ogni inserzione (in *ins_terra()*), ricordarsi di salvare in *ultima_terra* il puntatore all'ultima terra inserita (altrimenti si deve riscorrere la lista fino all'ultima terra quando si va ad inserire una nuova terra). Un esempio di lista dinamica con 3 terre è dato in Fig. 1.

A seconda del progetto che si vuole realizzare (**A** o **B**), si deve implementare un numero di funzioni differente:

Progetto A (fino a 2.5 punti). Chi svolge questo progetto deve solamente creare il percorso, quindi solo le funzioni e le strutture dati necessarie per i punti 1 e 3 in *progexam.c* (ignorare ogni funzione e struttura dati per muovere e rappresentare Oberon). Le funzioni da realizzare sono quindi le seguenti.

a) La funzione *crea_percorso()* viene richiamata da *progexam.c* e serve per chiedere al giocatore le operazioni da effettuare sul percorso: esse corrispondono alle funzioni (anche qui stampare un menu e leggere la risposta da tastiera) *ins_terra()*, *canc_terra()*, *stampa_percorso()*, e *chiudi_percorso()*. Tutte queste funzioni devono essere definite *static*, in quanto non devono essere chiamate da fuori *mylib.c*

   1) Inserimento di una terra in fondo alle terre già create (*ins_terra()*). Essa crea la nuova terra in memoria dinamica (*malloc()*), la inserisce nella lista modificando il valore del puntatore *terra_successiva* dell'ultima terra della lista con il risultato della *malloc()*, e aggiorna *ultima_terra* con il risultato della *malloc()*. Tutti i campi della nuova terra vengono inseriti da tastiera.
   2) Cancella l'ultima terra inserita nel percorso (*canc_terra()*). Liberare la memoria occupata dall'ultima terra ed aggiornare *ultima_terra* di conseguenza. Ricordarsi di trattare il caso particolare in cui nella lista non è presente nessuna terra.
   3) Stampa i campi di tutte le terre create fino a quel momento (*stampa_percorso()*).

4) Fine della creazione del percorso *chiudi_percorso()*. Ci si ricorda che il percorso è stato terminato (per esempio settando una variabile globale e statica da 0 a 1), e si esce anche dalla funzione *crea_percorso()*.

Si devono implementare i seguenti vincoli durante l'inserzione di una terra: *villaggio* non può contenere mostri, *scheletro* non può stare su *palude*, *orco* non può stare su *deserto*, *lupo* può stare solo su *foresta* o *pianura*, il tesoro in *villaggio* non può essere superiore a 10 monete d'oro. La prima terra del percorso non può contenere mostri. Gestire i vincoli chiedendo al giocatore di inserire nuovamente i campi della terra in questione.

Chi sceglie questo progetto controlli anche i paragrafi seguenti: deve essere anche implementata la funzione *termina_gioco()* come specificato nei "Commenti finali".

Progetto B (fino a 5 punti). Per questo progetto si deve implementare tutte le funzioni del progetto **A**, più le funzioni per muovere Oberon sul percorso e le strutture dati per rappresentarlo (punto 2 in *progexam.c*). Ricordarsi di inizializzare la struttura dati *static struct Oberon oberon* prima di azionarlo: Oberon inizia il gioco con 10 monete d'oro, 5 punti ferita, e 2 incantesimi, 1 pozione guaritrice. Le funzioni da realizzare sono quindi le seguenti.

a) La funzione *muovi_Oberon()* viene richiamata da *progexam.c* (punto 2) e serve per chiedere al giocatore le azioni che Oberon vuole compiere: esse corrispondono alle funzioni (anche qui stampare un menu e leggere la risposta da tastiera) *avanza()*, *prendi_tesoro()*, *usa_pozione*, *combatti()*, e *distruggi_terra()* (tutte definite come *static* in *mylib.c*). La funzione *muovi_Oberon()* ritorna subito al menu stampando un messaggio di avvertimento, in caso il percorso non sia stato ancora chiuso con *chiudi_percorso()* (vedi progetto **A**): il percorso deve essere chiuso prima di azionare Oberon.

   1) Oberon ad ogni momento si trova su una terra del percorso (ricordarsi la posizione con un puntatore a *struct Terra* e scorrere il percorso di conseguenza). Questa funzione serve ad avanzare di una terra (*avanza()*).
   2) La funzione *prendi_tesoro()* serve per mettere in *borsa_oro* tutto il tesoro presente sulla terra su cui si trova.
   3) La funzione *usa_pozione* ristabilisce i punti ferita di Oberon a 5, consumandola per sempre.
   4) La funzione *combatti()* è utilizzata per combattere il mostro presente sulla terra sulla quale Oberon si trova. Oberon può utilizzare quando vuole uno dei suoi incantesimi (consumandolo) uccidendo immediatamente il mostro, altrimenti può attaccare (un menu anche qui con le due opzioni, e lettura della scelta da parte del giocatore). Se generando un numero casuale (utilizzare Fig. 3) ottiene più di 60, allora sottrae 3 punti ferita al mostro. Se il mostro non muore dopo l'attacco, esso contrattacca: se generando un numero casuale ottiene più di 50 (ancora Fig. 3), sottrae un numero di punti ferita come riportato in seguito. Il combattimento prosegue (un attacco a testa) fino a quando uno dei due combattenti muore. Se Oberon muore, il gioco termina.
   5) Con *distruggi_terra* Oberon può distruggere la terra successiva a quella su cui si trova con una potente magia; questa funzione può essere chiamata una sola volta in tutto il gioco. Modificare la lista delle terre come in Fig. 2, ricordandosi di mantenere il percorso collegato.

Si devono implementare anche le seguenti regole di gioco. Lo zaino di Oberon non può trasportare più di 500 monete. Per raccogliere il tesoro di una terra, si deve prima sconfiggere eventuali mostri (se presenti). Un drago deve essere ucciso in combattimento per poter poi avanzare alla terra successiva (gli altri mostri no). *lupo* ha 1 punto ferita, *scheletro* ha 2 punti ferita, *orco* ha 3 punti ferita, e *drago* ha 5 punti ferita. Ogni mostro sottrae un numero di punti ferita equivalente, in caso di attacco riuscito.

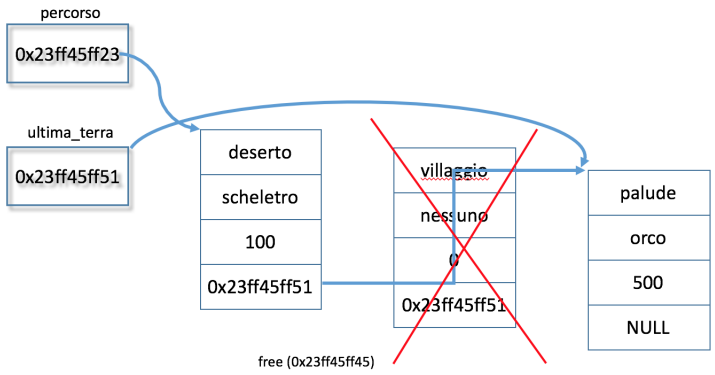

Figura 2: Oberon distrugge la terra successiva a quella in cui si trova (in *deserto*).

Per compilare. Includere `mylib.h` dentro `progexam.c` e dentro `mylib.c` (i.e. *#include "mylib.h"*). A questo punto si può compilare indipendentemente `progexam.c` e `mylib.c`, con rispettivamente i comandi `gcc -c progexam.c` e `gcc -c mylib.c` (vengono generati rispettivamente i file `progexam.o` e `mylib.o`). Infine, per comporre i due file, linkarli con `gcc -o progexam progexam.o mylib.o`. Aggiungere sempre i flag *-std=c11 -Wall* (per esempio `gcc -c progexam.c -std=c11 -Wall`), per essere sicuri di seguire lo standard *C 2011* e farsi segnalare tutti i *warning* rispettivamente.

Commenti finali. Ricordarsi di liberare tutto lo spazio non più utilizzato con *free()*, prima di uscire dal programma. Questo viene fatto quando si chiama la funzione *termina_gioco()*, scorrendo tutte le terre del percorso e facendo *free* su di esse. Chi esegue il progetto in singolo (e non in gruppo max. 3), non deve implementare la funzione 5 (progetto **B**). Se sceglie il progetto **A**) deve invece implementare tutte le funzioni.

```c
#include <stdio.h>
#include <stdlib.h> // Da includere per utilizzare rand() e srand()
#include <time.h> // Da includere per utilizzare time()

int main () {
  time_t t;

  /* Inizializza il generatore di numeri casuali utilizzando il tempo attuale */
  srand((unsigned) time(&t));  // Funzione da chiamare una volta sola nel programma

  /* Ritorna un numero tra 0 e 99 */
  printf("%d\n", rand() % 100); // Chiamare quando si ha bisogno di un numero random
}

```

Figura 3: Esempio di come funziona la generazione di un numero casuale.