28 Novembre 2022

Programmazione B
Ingegneria e Scienze Informatiche - Cesena
A.A. 2022-2023

Elaborato 9

Data di sottomissione: entro le 20 del 11 Dicembre 2022.
Formato di sottomissione: un file compresso con nome `elaborato9.zip`, contenente un unico file sorgente con nome `snake.c`
Codeboard: https://codeboard.io/projects/131852/

Specifiche:

- Sviluppare funzioni di libreria per poter gestire l'**oggetto serpente** nel gioco *snake*.
- Viene fornita l'implementazione dell'intero gioco, tranne l'implementazione della libreria `snake.c`. L'implementazione fa uso della libreria `curses`.
- I prototipi delle funzioni da implementare sono dichiarati nell'header `snake.h` e allegati alle specifiche.
- Il serpente è rappresentato tramite una **lista concatenata**.
- Il serpente è rappresentato tramite una struttura contenente:
  - un puntatore ad una **lista concatenata** contenente le posizioni del corpo del serpente (indici in una matrice bidimensionale),
  - la lunghezza attuale del serpente.
- Il mondo in cui si sposta il serpente è uno spazio **toroidale** rappresentato da una matrice di dimensione `rows`×`cols`.
- Le posizioni attuali del corpo del serpente sono salvate nella lista concatenata.
- Il serpente cresce di lunghezza aggiungendo nodi alla lista concatenata.

- Per semplificare le implementazioni, è possibile evitare di gestire gli eventuali casi di allocazione non riuscita di memoria: le dimensioni del gioco rendono alquanto improbabile che si verifichi questa possibilità.

Vincoli:

- Le implementazioni devono aderire perfettamente ai prototipi e alle specifiche fornite.
- Le eventuali funzioni di utility della libreria e variabili globali devono essere *nascoste* all'esterno.

Descrizione dettagliata delle funzioni:

- `snake_create()`: crea la struttura che rappresenta un serpente di lunghezza 1, posizionato un una qualsiasi cella in una matrice di dimensione `rows`×`cols`.
- `snake_kill()`: distrugge la struttura serpente (i.e. libera la memoria allocata dinamicamente).
- `snake_increase()`: aggiunge una nuova testa al serpente in una direzione specificata (up, down, right, left) rispetto alla testa attuale. Ad esempio, partendo dal set di coordinate

  {(1,1), (1,2), (2,2), (2,3)}

  dove assumiamo che la testa sia nella posizione (1,1), il nuovo set di coordinate in direzione `down` è equivalente al seguente set di coordinate

  {(2,1),(1,1), (1,2), (2,2), (2,3)}
- `snake_decrease()`: rimuove un determinato numero di parti del serpente, a partire dalla coda. Ad esempio, la rimozione di una singola posizione nel set di coordinate

  {(1,1), (1,2), (2,2), (2,3)}

  produce la seguente lista

  {(1,1), (1,2), (2,2)}
- `snake_move()`: sposta il serpente in una direzione (up, down, right, left). Tale funzione produce un nuovo set di posizioni per l'oggetto serpente equivalente al set di coordinate ottenuto con la seguente procedura:

1. aggiungiamo una nuova testa nella direzione `dir` rispetto alla vecchia testa,
2. rimuoviamo la posizione in coda

- `snake_reverse()`: inverte il serpente (i.e. la coda diventa la testa e viceversa). Ad esempio, una lista con posizioni

  {(1,1), (1,2), (2,2), (2,3)}

  dopo una chiamata alla `snake_reverse()` diventa

  {(2,3), (2,2), (1,2), (1,1)}

- `snake_head()`: ritorna la posizione attuale della testa del serpente.
- `snake_body()`: ritorna la posizione attuale di una parte del corpo del serpente, dove si assume che la testa sia in posizione 0.
- `snake_knotted()`: verifica che il serpente non si sia *annodato*. Per verificare se il serpente è annodato è sufficiente verificare che le coordinate della testa non siano ripetute nella lista.
- `snake_save()`: salva le coordinate del serpente in un file. Il formato di salvataggio è libero.
- `snake_load()`: carica le coordinate del serpente da un file.

Suggerimenti:

- Un aspetto da considerare prima di passare all'implementazione è dove posizionare la testa del serpente nella lista concatenata: in testa o coda?
- Come sono calcolate le nuove posizioni rispetto alle direzioni `up, down, left, right`? Ricordiamo che il mondo è toroidale di dimensione `rows`$\times$`cols`: possiamo attraversare i bordi per sbucare dalla parte opposta.
  - `up`: la posizione (i, j) diventa $((i - 1 + \texttt{rows}) \ \% \ \texttt{rows}, j)$
  - `down`: la posizione (i, j) diventa $((i + 1 + \texttt{rows}) \ \% \ \texttt{rows}, j)$
  - `left`: la posizione (i, j) diventa $(i, (j - 1 + \texttt{cols}) \ \% \ \texttt{cols})$
  - `right`: la posizione (i, j) diventa $(i, (j + 1 + \texttt{cols}) \ \% \ \texttt{cols})$

```c
#ifndef SNAKE_H
#define SNAKE_H

enum direction {UP, DOWN, LEFT, RIGHT};

struct position {
 unsigned int i;
 unsigned int j;
};

struct body {
 struct position pos;
 struct body *next;
 struct body *prev;
};

struct snake {
 unsigned int rows;
 unsigned int cols;
 unsigned int length;
 struct body *body;
};

/* Creates a snake's head in a world of size rows*cols */
struct snake *snake_create(unsigned int rows, unsigned int cols);

/* Destroys the snake data structure. */
void snake_kill(struct snake *s);

/* Returns the (coordinates of the) snake's head. */
struct position snake_head(struct snake *s);
/*
 * Returns the (position of the) i-th part of the snake body.
 * Position 0 is equivalento the snake's head. If the snake
 * is shorter than i, the position returned is not defined.
 */
struct position snake_body(struct snake *s, unsigned int i);

/*  Returns 1 if the snake crosses himself, 0 otherwise. */
int snake_knotted(struct snake *s);
/*
 * Moves the snake one step forward in the dir direction
 * into a toroidal world of size rows*cols.
 */
void snake_move(struct snake *s, enum direction dir);
/*
 * Reverses the snake.
 * The tail of the snake is now the new head.
 */
void snake_reverse(struct snake *s);
/*
 * Increases the snake length.
 * This is equivalent to:
 * - add a new head in the dir direction wrt the old head.
 */
void snake_increase(struct snake *s, enum direction dir);

/* Removes from the tail of the snake decrease_len parts.*/
void snake_decrease(struct snake *s, unsigned int decrease_len);

/* Saves the snake into the filename. */
void snake_save(struct snake *s, char *filename);

/* Loads the snake from filename */
struct snake *snake_read(char *filename);

#endif
```