21 Novembre 2016

Programmazione M-Z
Ingegneria e Scienze Informatiche - Cesena
A.A. 2016-2017

Elaborato 8

Data di sottomissione: entro la mezzanotte del 4 Dicembre 2016.
Formato di sottomissione: un file compresso con nome `Elaborato8.zip`, contenente un unico file sorgente con nome `snake.c`

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 struttura contenente
  - un *array dinamico di coordinate*,
  - la lunghezza attuale dell'array.
- Le funzioni di libreria in `snake.c` si occupano di:
  - creare una struttura serpente (i.e. allocare dinamicamente memoria per la struttura),
  - distruggere (i.e. liberare la memoria allocata dinamicamente) una struttura serpente,
  - spostare il serpente in una direzione (up, down, right, left),
  - aggiungere una nuova testa al serpente in una direzione specificata (up, down, right, left) rispetto alla testa attuale,
  - dimezzare la dimensione del serpente, rimuovendo la parte in coda.
  - ritornare le coordinate della testa attuale del serpente,

  - verificare che il serpente non si sia *annodato* (i.e. che il vettore non contenga ripetizioni delle stesse coordinate).

- Le funzioni di libreria prendono in input un puntatore alla struttura serpente `s` e la lunghezza attuale del serpente `length`:
  - Le coordinate attuali del serpente sono salvate nelle celle

    `s->coord[0], ..., s->coord[length-1]`.

  - Le funzioni conoscono la lunghezza effettiva dell'array `s->coord` (salvata nel campo `s->max_length`) e si devono occupare di ri-allocare opportunamente l'array qualora qualche operazione di incremento della dimensione del serpente non possa essere effettuata nella dimensione attuale dell'array.

- 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.

- Differenze rispetto alla versione `1.0` (in sintesi):
  - Il serpente può crescere *illimitatamente*: la dimensione massima dipende unicamente dalla dimensione della matrice di gioco.
  - Mangiare un frutto cattivo ha come effetto il dimezzamento della lunghezza serpente.

Vincoli:

- Le implementazioni devono aderire perfettamente ai prototipi e alle specifiche fornite.

- Le eventuali funzioni di utility della libreria devono essere *nascoste* all'esterno.

Suggerimenti:

- Un aspetto da considerare prima di passare all'implementazione è dove posizionare la testa del serpente nell'array.

- La funzione di di spostamento nella direzione `dir` produce un nuovo set di coordinate per l'oggetto serpente. Il nuovo set di coordinate è 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 coordinata in coda.

- Per verificare se il serpente è annodato è sufficiente verificare che le coordinate della testa non siano ripetute nell'array.
- Anche se è alquanto improbabile che si verifichi qualche problema, prestare attenzione all'operazione di riallocazione dell'array.

```c
#ifndef SNAKE_H
#define SNAKE_H

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

struct coord {
 unsigned int x;
 unsigned int y;
};

struct snake {
 unsigned int max_length;
 struct coord *coord;
};

/*
 * Creates a snake's head at coordinates (x,y);
 */
struct snake *snake_create(unsigned int x, unsigned int y);

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

/*
 * Returns the (coordinates of the) snake's head.
 */
struct coord snake_head(struct snake *s, unsigned int length);

/*
 * Returns 1 if the snake crosses himself, 0 otherwise.
 */
int snake_knotted(struct snake *s, unsigned int length);

/*
 * Moves the snake one step forward in the dir direction.
 */
void snake_move(struct snake *s, unsigned int length,
                                          enum direction dir);

/*
 * 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, unsigned int length,
                                          enum direction dir);
/*
 * Decreases the snake length by half.
 *
 * This is equivalent to:
 * - remove ceil(length/2) coordinates from the tail of the snake.
 */
void snake_decrease(struct snake *s, unsigned int length);

#endif
```