Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 26 Ottobre 2006

Riepilogo di concetti vari del linguaggio C che possono tornare utili

- Gli operatori logici sono `&&` per l'AND, `||` per l'OR, `!` per il NOT.
- `&&` e `||` hanno la *valutazione cortocircuitata*: Si garantisce che la valutazione del loro primo operando viene eseguita per prima. La valutazione del secondo non è eseguita affatto se la valutazione del primo è sufficiente a determinare il valore totale dell'espressione.
- L'operatore `%` fornisce il resto della divisione intera: esempio: `11 % 2 == 1`.
- Le definizioni di macro (anche quelle con parametri) non devono essere terminate da `;`
- In C non esiste un tipo `boolean`. Per memorizzare valori booleani si utilizzi il tipo `char`.

Esercizio 1: tris

Si tratta di una semplice implementazione del gioco del *tris*. Su una tavola di dimensioni 3×3 due giocatori si alternano nel posizionare le proprie pedine, una per ogni mossa. Vince chi ne mette 3 in fila, in una colonna, o in una riga, o in una delle due diagonali. Se tutte le caselle della tavola sono occupate e nessuno ha ancora vinto, la partita finisce in pareggio.

Ogni casella della tavola è individuata da una coppia di interi (assumiamo che siano `short`) (i, j) dove $0 \leq i, j \leq 2$. La coppia (i, j) costituisce la *posizione* della casella. La casella in posizione (i, j) è la casella nella i-esima riga e nella j-esima colonna.

Si *rappresenti* la tavola con un **array di 9 char**:

```
char tavola[9];
```

I primi 3 elementi dell'array `tavola` rappresentano le 3 caselle della prima riga della tavola, i secondi 3 rappresentano le 3 caselle della seconda riga, gli ultimi 3 rappresentano le 3 caselle della terza riga.

Si *consiglia* di implementare una *macro con parametri*:

```
#define posto(i,j)
```

(o se non siete confidenti con le macro, una funzione `short posto(short i, short j)`) tale che `posto(i,j)` sia l'indice dell'elemento dell'array `tavola` in cui è rappresentata la casella di posizione (i, j). Ad esempio devono valere le seguenti eguaglianze: `posto(1,1) == 4, posto(2,2) == 8, posto(0,0) == 0, posto(1,2) == 5`.

Ogni casella può essere libera, o essere occupata da una pedina del giocatore 1, rappresentata dal carattere `'*'`, o essere occupata da un pedina del giocatore 2, rappresentata dal carattere `'o'`.

Quindi si *osserva* che lo stato di ogni casella della tavola è completamente specificato assegnando all'elemento corrispondente nell'array `tavola` un valore $x \in \{0, 1, 2\}$ (dunque ogni possibile valore di x è memorizzabile già in un `char`).

Si *richiede* di implementare le seguenti funzioni:

- `void stampatavola(char *tavola)`

  Stampa a video lo stato attuale della tavola `tavola` come nell'esempio seguente:
  Supponiamo che il giocatore 1 abbia occupato solo le due caselle nelle posizioni $(1, 1)$ e $(0, 2)$, e che il giocatore 2 abbia occupato solo le due caselle nelle posizioni $(1, 0)$ e $(2, 0)$, allora la tavola andrà visualizzata come segue:

  ```
  -------
  | | |*|
  -------
  |o|*| |
  -------
  |o| | |
  -------
  ```

- `void puliscitavola(char *tavola)`

  Prepara la tavola per iniziare una nuova partita, vale a dire, libera ogni casella della tavola.

- `void eseguimossa(char *tavola, char giocatore, short i, short j)`

  Se la casella in posizione (i, j) è libera, viene occupata da una pedina del giocatore `giocatore`, altrimenti non compie alcuna operazione.

  Non si richiede che `eseguimossa` debba necessariamente controllare che (i, j) sia la posizione di una casella (vale a dire, `eseguimossa` non è tenuta a controllare che $0 \leq i, j \leq 2$).

- `char mossalegale(char *tavola, short i, short j)`

  Restituisce `1` se $0 \leq i, j \leq 2$ e la casella di posizione (i, j) è libera. Restituisce `0` altrimenti.

- `void inputmossa(char *tavola, char giocatore, ___, ___)`

  Stampa sul video il messaggio:
  `Giocatore` *g*`:  muovi:`
  dove g è 1 se `giocatore == 1`, 2 altrimenti.

  Successivamente, legge dalla tastiera due valori interi i e j, separati da caratteri di spaziatura, che rappresentano la mossa del giocatore g, vale a dire la richiesta di posizionare una delle sue pedine nella casella di posizione (i, j). L'informazione sui valori i e j letti va conservata e passata alla funzione chiamante (specificamente la funzione `partita`, vedi sotto) in modo che tale funzione possa di seguito invocare `eseguimossa(tavola, giocatore, i, j);`

  A tale scopo è necessario *simulare il passaggio per riferimento* delle due variabili `short i,j` a `inputmossa`. Fa parte dell'esercizio il compito di stabilire i tipi da attribuire al terzo e quarto parametro di `inputmossa` al fine di simulare il passaggio per riferimento.

  Si ricorda che le specifiche da inserire nella stringa formato in una chiamata del tipo `scanf(formato, ...)` sono:

  - `%d` per leggere un intero e memorizzarlo in un `int`.
  - `%hd` per leggere un intero e memorizzarlo in uno `short`.
  - `%c` per leggere un carattere e memorizzarlo in un `char`.

  A questo stadio dell'esercitazione non si richiede di eseguire alcun controllo sulla correttezza dell'input.

- `char havinto(char *tavola)`

  Restituisce 1 se lo stato attuale della tavola contiene un tris del giocatore 1, 2 se lo stato attuale della tavola contiene un tris del giocatore 2, 0 se nessuno dei due giocatori ha ancora realizzato un tris.

- `char tavolapiena(char *tavola)`

  Restituisce 1 se la tavola non ha caselle libere, 0 altrimenti.

- `void partita(void)`

  Lancia una partita. Vale a dire:

  1. Prepara la tavola nello stato iniziale.
  2. Stampa la tavola.
  3. Alterna le mosse dei due giocatori fino a quando uno dei due vince, oppure la tavola si riempie senza vincitori.
  4. Dopo ogni mossa di ognuno dei giocatori, stampa la tavola aggiornata.
  5. Se alla fine della partita il giocatore 1 ha vinto, stampa il messaggio: `Ha vinto il giocatore 1!`, se invece ha vinto il giocatore 2, stampa: `Ha vinto il giocatore 2!`, altrimenti stampa: `Partita conclusa in pareggio`.

- La funzione `main` si limita a lanciare una partita.

Tutte le variabili usate devono essere *locali*, vale a dire, *dichiarate* (concetto che in questi casi coincide con *definite*) *nel corpo* di qualche funzione. In particolare, la dichiarazione dell'array `char tavola[9]` deve essere locale alla funzione `partita`.

Supplementi

Da svolgere nell'ordine specificato.

- Nella funzione `inputmossa` si controlli che ogni mossa immessa da un giocatore sia una `mossalegale`. In caso contrario si rigetti l'input attuale, stampando il messaggio `Input scorretto, riprovaci.`, e successivamente si richieda un nuovo input (stampa sul video del messaggio `Giocatore` *g*`: muovi:`, e passi successivi di `inputmossa`.)

- Nella funzione `inputmossa` si controlli che i dati immessi dal giocatore corrispondano effettivamente a due interi separati da caratteri di spaziatura. In caso contrario si rigetti l'input attuale, comportandosi in seguito come al punto precedente.

  Si ricordi a tale scopo che la funzione di libreria `scanf` restituisce il numero di argomenti della sua stringa formato che è riuscita ad abbinare all'input. Nel nostro caso la chiamata di `scanf` deve restituire `2` se i dati immessi corrispondono a due interi.

  Può inoltre essere utile scartare tutto il resto della riga di input attualmente a disposizione, non appena si riscontra un errore, in modo da poter ripartire da un input *pulito* per le successive letture. Per svuotare l'input si inserisca la seguente definizione di macro:

  ```
  #define svuotainput()     while(getchar() != '\n')
  ```

  e si richiami `svuotainput()` ogni qual volta sia necessario.

- Modificare la funzione `main` in modo che si comporti come di seguito descritto:

  1. Lancia una `partita`
  2. Alla fine della partita stampa il messaggio: `Un'altra partita? (s/n)`.
  3. Legge dall'input un singolo carattere.
  4. Se tale carattere è `'s'`, allora itera dal punto 1, altrimenti termina l'esecuzione del programma.

  (Anche in questo caso può essere utile svuotare l'input prima di lanciare una nuova partita.)

- Generalizzare il gioco come segue:
  - Introdurre una macro `#define LATO 3`
  - Adattare il resto del programma in modo che reimpostando la macro `LATO` a un valore k diverso da `3`, si ottenga una versione del gioco su tavola $k \times k$, in cui per vincere bisogna allineare k pedine.