CORSO DI FONDAMENTI DI INFORMATICA

Prof. Maristella Matera - A.A. 2017 / 2018

Laboratorio di Programmazione in C – Laboratorio 3

Esercizio 1 – Sudoku

Si realizzi un programma che verifichi la corretta soluzione della griglia di un Sudoku.

Un Sudoku è una griglia 9x9 che rispetta queste caratteristiche:

- tutti i valori devono essere singoli numeri tra 1 e 9
- su ciascuna delle 9 **righe** non devono esserci valori ripetuti
- su ciascuna delle 9 **colonne** non devono esserci valori ripetuti
- in ciascuno dei 9 **blocchi 3x3** che compongono la griglia non devono esserci valori ripetuti

L'esercizio è da risolvere svolgendo un livello dopo l'altro, secondo questa sequenza:

Livello 1 – Utilizzare il file `sudoku_template.c` fornito assieme a questo testo come punto di partenza: il file contiene la dichiarazione di due matrici Sudoku da verificare (una corretta e l'altra sbagliata). Il programma deve verificare la validità di queste due matrici e stampare a video il risultato del controllo. In questo livello il programma va scritto **senza** utilizzare le funzioni (quindi completando solo il `main()` già esistente).

Livello 2 – Modificare il programma già scritto usando le funzioni per dividere il problema assegnato in sottoproblemi.

Livello 3 – Modificare il programma già scritto in modo che la matrice da verificare venga acquisita numero per numero, e che venga eseguito il controllo di validità già in fase di inserimento.

Livello 4 – Modificare il programma già scritto in modo che la matrice da verificare venga letta dal file di testo `sudoku.txt` fornito assieme a questo testo.

Esercizio 2 - Mosse su matrice (tratto dal TdE di Novembre 2015)

Sia data una matrice `M` di `N×N` numeri interi positivi (`N` costante) e un array `A` di `K` elementi (`K` costante) organizzato in base alla seguente dichiarazioni di tipo:

```
#define K
...
typedef struct {
     int mossaRiga;
     int mossaColonna;
} Mossa;

Mossa A[K];
```

Gli elementi di `A` rappresentano spostamenti all'interno della matrice `M` a partire dal suo primo elemento `M[0][0]`. I campi `mossaRiga` e `mossaColonna` possono assumere valori nell'insieme {-1, 0, 1}. In particolare:

- Il valore -1 indica uno spostamento verso la riga (colonna) precedenti rispetto alla posizione corrente
- Il valore 0 indica di non spostarsi
- Il valore 1 indica uno spostamento verso la riga (colonna) successiva.

Per esempio, se i valori di un elemento dell'array, `A[i]`, sono

```
<A[i].mossaRiga = 0, A[i].mossaColonna = 1>
```

allora rispetto all'elemento corrente è necessario rimanere nella stessa riga e muoversi di una colonna a destra.

Si scriva un programma che, una volta inizializzate le strutture dati M e A, calcoli e restituisca come valore di ritorno la somma degli elementi della matrice visitati seguendo le mosse memorizzate in A. Nel caso di mosse che portano a posizioni non lecite (cioè oltre i limiti imposti dalle dimensioni della matrice) il programma restituisce il valore -1 e termina immediatamente.

Esercizio 3 - Fusione di array ordinati con Bubblesort

Scrivere un programma che:

- chiede due volte all'utente di inserire N valori interi, e li salva in due array di dimensione N;
- ordina quindi ciascun array utilizzando il metodo Bubble Sort;
- effettua la fusione dei due array, in modo che l'array risultante mantenga l'ordinamento;
- stampa il vettore dopo la fusione.

L'algoritmo Bubble Sort confronta ogni coppia di elementi adiacenti del vettore. Se due elementi sono nell'ordine sbagliato, essi vengono invertiti (swap). Come risultato, a ogni passo l'elemento più grande non ancora ordinato si sposta verso la posizione più alta dell'array. Al passo p, gli ultimi p-1 elementi dell'array saranno nella loro posizione definitiva (non è necessario controllarli di nuovo). L'algoritmo continua a scorrere tutta la lista finché non vengono più eseguiti scambi, situazione che indica che la lista è ordinata.

Implementare l'algoritmo scomponendo il problema in sottoproblemi mediante l'uso di funzioni. Creare una funzione per eseguire l'ordinamento bubblesort di un vettore, una funzione per eseguire lo swap di due elementi, e una funzione per fondere due array in un unico vettore.

Valutare inoltre la complessità dell'algoritmo di ordinamento contando il numero totale di confronti e il numero totale di swap eseguiti e stampare questi valori.