LABORATORIO FONDAMENTI DI INFORMATICA 9-10 DICEMBRE 2013

ESERCIZIO 1

Definire il tipo di dati

```
typedef float t_matrice[DIM][DIM];
```

e implementare la funzione

```
void calcola(t_matrice matr_media, int matr_orig[DIM][DIM])
```

Tale funzione analizza la matrice di numeri interi "`matr_orig`" passata come parametro e calcola e inserisce dei valori reali nella matrice "`matr_media`" anch'essa passata come parametro. In tale matrice ciascun elemento `[i][j]` dovrà essere un numero reale uguale alla media aritmetica (somma degli elementi diviso numero degli elementi) dei valori presenti nella matrice "`matr_orig`" nelle caselle adiacenti all'elemento `[i][j]` (senza coinvolgere l'elemento `[i][j]`). Le caselle "adiacenti" a una determinata casella `[i][j]` sono al massimo 8 ma sono di meno quando `[i][j]` è su uno dei bordi della matrice.

Per verificare se una casella adiacente esiste, ovvero se le sue coordinate `[i+h][j+k]` con -1<=`k`<=1 e -1<=`h`<=1 sono interne alla matrice, realizzare la seguente funzione:

```
int dentro(int x, int y)
```

Tale funzione ritorna `1` se le coordinate `x` e `y` sono interne alla matrice (cioè se `x` e `y` sono indici validi >= 0 e < `DIM`), altrimenti ritorna `0`.

Per calcolare la media relativa a ciascun elemento `[i][j]`, occorre richiamare l'ulteriore funzione da implementare

```
float media(int matr_orig[DIM][DIM], int i, int j)
```

Per gli elementi sui bordi, la media va fatta utilizzando solo gli elementi adiacenti che appartengono alla matrice (interni alla matrice, le cui coordinate fanno ritornare 1 alla funzione "`dentro`").

Nella funzione `main` del programma occorre dichiarare le due matrici "`matr_media`" e "`matr_orig`", riempire a piacere "`matr_orig`" e richiamare la funzione "`calcola`" per poi stampare a video le due matrici

```
Matrice:                    Matrice delle medie:
 1  2  3                     3.67  3.80  4.33
 4  5  6                     4.60  5.00  5.40
 7  8  9                     5.67  6.20  6.33
```

ESERCIZIO 2

Seguendo le indicazioni, scrivere un programma che simuli la gestione del magazzino di un negozio di ferramenta.

1) Definire un enumerato per le tipologie di articoli che il magazzino può contenere:

```
typedef enum {FERRAMENTA, CASALINGHI, GIARDINAGGIO} tipologia;
```

2) Definire il seguente tipo di dato per rappresentare un articolo di ferramenta:

```
typedef struct {
   tipologia tipo;
   int qta_attuale;
   int soglia_riordino;
   char note[100];
} elem_magazzino;
```

3) Implementare la funzione "`aggiorna_articolo`" che ritorna un numero intero e riceve come parametri una variabile "`articolo`" di tipo "`elem_magazzino`" (che deve essere modificata dalla funzione, attenzione alla modalità di passaggio) e un numero intero

“qta_da_vendere” che rappresenta la quantità di articoli che si ha intenzione di vendere. Nel caso in cui l'articolo passato come parametro sia presente in quantità sufficiente (cioè “qta_attuale” >= “qta_da_vendere”) occorre decrementare il campo “qta_attuale”.
Nel caso in cui invece la quantità sia insufficiente, occorre riempire il campo “note” con la stringa “qta_insufficiente” e ritornare -2.
Nel caso in cui la quantità sia sufficiente ma la “qta_attuale” dopo il decremento scenda sotto la “soglia_riordino”, occorre riempire il campo “note” con la stringa “da riordinare” e ritornare -1.
Se la quantità è sufficiente e la quantità non scende sotto la soglia occorre ritornare 0.
4) Implementare la funzione “aggiorna_magazzino” che ritorna un numero intero e riceve come parametri un vettore “magazzino” di massimo DIM=30 celle di tipo “elem_magazzino” e un numero intero “n” che indica quante celle sono effettivamente riempite con elementi significativi.
La funzione deve analizzare uno a uno gli “n” articoli del vettore e per ciascuno di essi, nel caso in cui siano di tipo CASALINGHI occorre invocare la funzione “aggiorna_articolo” passando come valore per il parametro “qta_da_vendere” un numero casuale
qta_da_vendere = rand() % 10 + 1;
La funzione deve ritornare la somma delle quantità rimanenti di tutti gli articoli di tipo CASALINGHI.
5) Nella funzione main del programma occorre creare un array di massimo DIM celle di tipo “elem_magazzino” e riempirlo in maniera casuale con istruzioni del tipo sottoindicato per poi richiamare la funzione “aggiorna_magazzino” e stampare a video degli opportuni messaggi per indicare lo stato del magazzino.

```
int n = rand() % DIM; //per stabilire il numero di articoli
for (i = 0; i < n; i++) {
  m[i].tipo = rand() % 3; //corrispondenza enum <-> numero intero
  m[i].qta_attuale = rand() % 10;
  m[i].soglia_riordino = rand() % 5;
  strcpy(m[i].note, “”);
}
```

```
Generazione di un magazzino con 7 articoli...
Generato articolo #0 tipo=1 qta_attuale=4 soglia_riordino=3 note=””
…
Generato articolo #2 tipo=2 qta_attuale=3 soglia_riordino=1 note=””
…
Generato articolo #5 tipo=1 qta_attuale=9 soglia_riordino=0 note=””
…
Aggiornamento del magazzino...
Controllo articolo #0...
Articolo #0 e' di tipo CASALINGHI. Tento di venderne 10.
Quantita' insufficiente... Note=”qta_insufficiente”
…
Controllo articolo #2...
Articolo #2 non e' di tipo CASALINGHI
…
```

```
Controllo articolo #5...
Articolo #5 e' di tipo CASALINGHI. Tento di venderne 1.
Vendita andata a buon fine. Note=""
…
Articoli di tipo CASALINGHI rimanenti = 16
```

ESERCIZIO 3

Si decide di adottare un metodo alternativo per memorizzare una generica matrice bidimensionale. Con questo metodo, invece che tramite un array bidimensionale, la matrice viene rappresentata tramite un array di elementi di questo tipo:

```
typedef struct {
     int riga;
     int colonna;
     int valore;
} t_elementi;
```

Si ipotizza che la matrice sia sparsa, ovvero che contenga un gran numero di zeri; per questo occorre memorizzare solo gli elementi con valore diverso da zero.

Realizzare un programma che stampi a video il prodotto elemento per elemento di due matrici (`DIM` x `DIM` con `DIM` = 5) rappresentate in questo modo "alternativo" implementando le seguenti funzioni:

1) `void matrice_completa(int matrice[DIM][DIM])`

Riempie una "normale" matrice passata come parametro con numeri casuali interi >= 0. Questa "normale" matrice verrà convertita nella nuova rappresentazione da un'altra funzione.

2) `int memorizza_matr(t_elementi elementi_matrice[DIM*DIM])`

Richiama la funzione "`matrice_completa`" per riempire una "normale" matrice di interi. La matrice "normale" deve essere letta per popolare l'array "`elementi_matrice`" memorizzando in esso solo gli elementi diversi da zero.

La funzione deve restituire il numero di elementi memorizzati in "`elementi_matrice`".

3) `int leggi_matr(t_elementi elementi_matrice[DIM*DIM], int n, int riga, int colonna)`

Questa funzione restituisce il `valore` presente nella "matrice" alla `riga` e `colonna` specificate. La ricerca del `valore` richiesto deve essere effettuata nel vettore di strutture "`elementi_matrice`". Se l'elemento cercato non esiste in tale vettore, significa che il valore originale inserito era zero. Il parametro "`n`" indica il numero di valori memorizzato nell'array.

3) `int calcolo()`

Dopo aver popolato due diversi array `m1` e `m2` di tipo `t_elementi` tramite la funzione "`memorizza_matr`", la funzione "`calcolo`" deve stampare a video il prodotto elemento per elemento delle due matrici rappresentate col metodo alternativo, utilizzando la funzione "`leggi_matr`" per leggerne i singoli elementi.

```
Matrice m1:          Matrice m2:          Matrice prodotto:
 7  9  3  8  0       3  9  8  6  5        21 81 24 48  0
 2  4  8  3  9       7  6  2  7  0        14 24 16 21  0
 0  5  2  2  7       3  9  9  9  1         0 45 18 18  7
 3  7  9  0  2       7  2  3  6  5        21 14 27  0 10
 3  9  9  7  0       5  8  1  4  7        15 72  9 28  0
```