Esercizio 1
(Matrici)

Valutazione delle temperature medie

- Una stazione meteorologica registra la temperatura esterna ogni ora, ogni griono per un mese intero, in una struttura dati apposita: una matrice bi-dimensionale di dimensioni fisiche di 31 righe (i giorni del mese) per 24 colonne (le ore del giorno).
- Realizzare alcune opportune funzioni che calcolino i seguenti dati:
  - Temperatura media di tutto il mese
  - Temperatura media giornaliera (per ogni giorno o per un giorno specifico???)
  - Temperatura media diurna (ore comprese tra le 7.00 e le 19.00), calcolata su tutto il mese
  - Temperatura media notturna calcolata su tutto il mese

Esercizio 1
(Matrici)

Valutazione delle temperature medie

- Nota: non tutti i mesi hanno 31 giorni… le funzioni quindi riceveranno come parametri la dimensione logica della martice…

- Si crei una funzione opportuna che inizializzi con valori casuali la matrice. A tal scopo si usi la funzione

```
int rand();
```

che restituisce un valore casuale compreso tra 0 e RAND_MAX (pari almeno a 32767) (stdlib.h)

Esercizio 1 - Soluzione
(Matrici)

```
#include <stdio.h>
#include <stdlib.h>

#define NUM_GIORNI 31
#define NUM_ORE 24

void initMatrice(int * m, int sizer, int sizec) {
   int i, j;

   for (i=0; i<sizer; i++) {
        for (j=0; j<sizec; j++) {
               // init all the values between 18 and 20
               m[i*sizec +j] = rand()%3 + 18;
        }
   }
}
```

Esercizio 1 - Soluzione
(Matrici)

```
void printMatrice(int * m, int sizer, int sizec) {
   int i, j;

   for (i=0; i<sizer; i++) {
        for (j=0; j<sizec; j++) {
               printf("%4d ", m[i*sizec +j]);
        }
        printf("\n");
   }
}
```

Esercizio 1 - Soluzione
(Matrici)

```
float tempMediaMese(int m[][NUM_ORE], int dimGiorni) {
   float result = 0;
   int i, j;

   for (i=0; i<dimGiorni; i++) {
        for (j=0; j<NUM_ORE; j++) {
                result = result + m[i][j];
        }
   }
   return result/(NUM_ORE*dimGiorni);
}
```

Esercizio 1 - Soluzione
(Matrici)

```
float tempMediaGiorno(int m[][NUM_ORE], int dimGiorni, int giorno) {
   float result = 0;
   int i;

   if (giorno < dimGiorni) {
        for (i=0; i<NUM_ORE; i++) {
                result = result + m[giorno][i];
        }
   }
   return result/(NUM_ORE);
}

void stampaTempMediaGiorni(int m[][NUM_ORE], int dimGiorni) {
   int i;

   for (i=0; i < dimGiorni; i++) {
        printf("Temperatura media giorno %d: %3.2f\n", i+1,
   tempMediaGiorno(m, dimGiorni, i));
   }
}
```

Esercizio 1 - Soluzione
(Matrici)

```
float tempMediaDiurna(int m[][NUM_ORE], int dimGiorni) {
   float result = 0;
   int i, j;

   for (i=0; i<dimGiorni; i++) {
        for (j=0; j<NUM_ORE; j++) {
               if (j>=7 && j<19)
                     result = result + m[i][j];
        }
   }
   return result/((NUM_ORE/2)*dimGiorni);
}
```

Esercizio 1 - Soluzione
(Matrici)

```
float tempMediaNotturna(int m[][NUM_ORE], int dimGiorni) {
   float result = 0;
   int i, j;

   for (i=0; i<dimGiorni; i++) {
        for (j=0; j<NUM_ORE; j++) {
               if (j<7 || j>=19)
                     result = result + m[i][j];
        }
   }
   return result/((NUM_ORE/2)*dimGiorni);
}
```

Esercizio 1 - Soluzione
(Matrici)

```
int main() {
   int temp[NUM_GIORNI][NUM_ORE];

   initMatrice(&temp[0][0], NUM_GIORNI, NUM_ORE);
   printMatrice(&temp[0][0], NUM_GIORNI, NUM_ORE);

   printf("Temperatura media mensile: %f\n", tempMediaMese(temp, 31));
   stampaTempMediaGiorni(temp, 31);
   printf("Temp. media diurna: %f\n", tempMediaDiurna(temp, 31));
   printf("Temp. media notturna: %f\n", tempMediaNotturna(temp, 31));

   system("PAUSE");

}
```

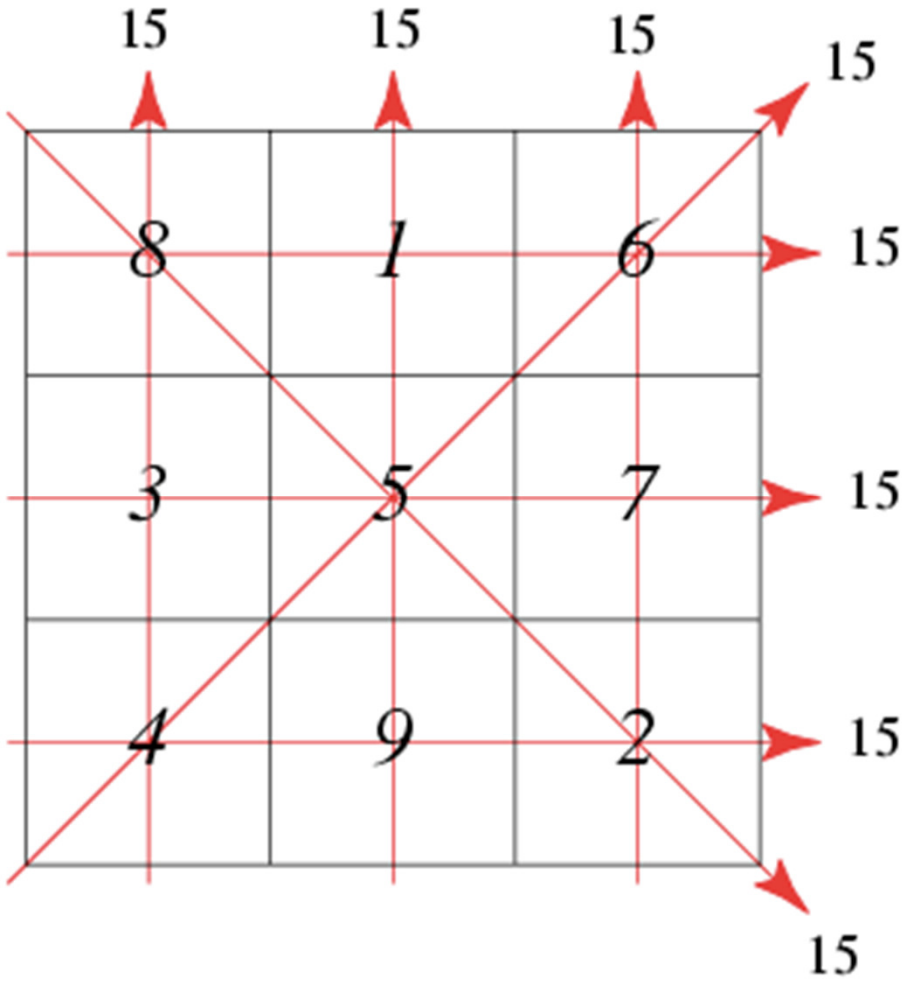
Esercizio 2
(Matrici)

Magic Square

- Realizzare una funzione che, presa in input una matrice quadrata, determini ***se è un quadrato magico***
- Un quadrato magico è una matrice NxN
  - I cui elementi sono TUTTI i numeri interi da 1 a N^2
  - Le somme degli elementi per tutte le righe, tutte le colonne e le diagonali sono uguali
    - Tale somma è detta “magic constant”

Esercizio 2
(Matrici)

Esempio di quadrato magico



Esercizio 2
(Matrici)

Linee guida

- Ragionare sempre a livelli di astrazione e decomporre la funzione in sotto-funzioni
- ***Quattro sotto-funzioni***
  1. Verifica che la matrice sia ben formata
  2. Verifica che la somma di ogni riga sia equivalente (e in caso affermativo, restituisce tale valore)
  3. Come 2, ma sulle colonne
     - NOTA: il calcolo è molto simile a quello del punto 2
  4. Come 2, ma sulle diagonali
- Una funzione che, data in input la matrice e la sua dimensione, invoca opportunamente le sotto-funzioni e restituisce dei codici differenziati
  - Nel caso in cui la matrice sia un quadrato magico, restiuisce anche la “magic constant”

Esercizio 2 - Soluzione
(Matrici)

```
#define N 3
typedef int matrice[N][N];
typedef enum{false, true} boolean;
typedef boolean covered[N*N];
boolean verifyMatrix(matrice m, int dim)
{
   covered c;
   int i, j;
   for(i = 0; i < dim*dim; i++)
       c[i] = false;
   …
```

Vettore che verifica la presenza, nella matrice, dei numeri da 1 a N^2

Se c[i] è true significa che il numero i+1 è presente nella matrice

Esercizio 2 - Soluzione
(Matrici)

```
…
    for(i = 0; i < dim; i++)
    {
         for(j = 0; j <  dim; j++)
         {
                 if(m[i][j] < 1 || m[i][j] > dim*dim)
                 {
                         return false;
                 }
                 if( c[ m[i][j] -1] == false)
                 {
                         c[ m[i][j] -1] = true;
                 }
                 else
                 {
                         return false;
                 }
         }
    }
    return true;
}
```

Controllo sull'intervallo dei valori [1,N^2]

I numeri non devono essere ripetuti

Esercizio 2 - Soluzione
(Matrici)

- Essendo il calcolo su righe e colonne molto simile, realizziamo ***un'unica funzione sufficientemente generica***
  - La somiglianza è dovuta al fatto che la matrice è quadrata
  - Un ulteriore parametro di questa funzione indica se il conteggio va effettuato sulle righe o sulle colonne

```
boolean verifyRows(matrice m, int dim, int* sum)
{
   return verify(m, dim, true, sum);
}

boolean verifyColumns(matrice m, int dim, int* sum)
{
   return verify(m, dim, false, sum);
}
```

Esercizio 2 - Soluzione
(Matrici)

```
boolean verify(matrice m, int dim, boolean verifyRows, int* sum)
{
    int i, j, partial;
    *sum = -1;
    for(i = 0; i < dim; i++)
    {
          partial = 0;
          for(j = 0; j < dim; j++)
          {
                  if(verifyRows)
                          partial = partial + m[i][j];
                  else
                          partial = partial + m[j][i];
          }
          if(*sum < 0)
                  *sum = partial;
          else if (*sum != partial)
                  return false;
    }
    return true; }
```

Inizializzazione della somma

Esercizio 2 - Soluzione
(Matrici)

```
boolean verifyDiagonals(matrice m, int dim, int* sum)
{
   int i, j, sum2 = 0;
   *sum = 0;
   for(i = 0; i < dim; i++)
       *sum = *sum + m[i][i];
   for(i = 0; i < dim; i++)
       sum2 = sum2 + m[i][dim-i-1];
   return (*sum == sum2);
}
```

Esercizio 2 - Soluzione
(Matrici)

Definizione dei codici di ritorno della funzione

```
#define RESULT int

#define NOT_WELL_FORMED 0;
#define ROWS_CHECK_FAILED 1;
#define COLUMNS_CHECK_FAILED 2;
#define DIAGONALS_CHECK_FAILED 3;
#define DIFFERENT_SUM 4;
#define MAGIC 5;
```

Esercizio 2 - Soluzione
(Matrici)

```
RESULT magicSquared(     matrice m, int dim,
                         int*magic_constant)
{
   boolean result;
   int sum, sum2;
   result = verifyMatrix(m, dim);
   if(!result)
       return NOT_WELL_FORMED;
   result = verifyRows(m, dim, &sum);
   if(!result)
       return ROWS_CHECK_FAILED;
   result = verifyColumns(m, dim, &sum2);
   if(!result)
       return COLUMNS_CHECK_FAILED;
   …
```

Esercizio 2 - Soluzione
(Matrici)

```
    …
    if(sum != sum2)
        return DIFFERENT_SUM;
    result = verifyDiagonals(m, dim, &sum2);
    if(!result)
        return DIAGONALS_CHECK_FAILED;
    if(sum != sum2)
        return DIFFERENT_SUM;
    *magic_constant = sum;
    return MAGIC;
}
```