Note

È considerato errore qualsiasi output non richiesto dagli esercizi.
È importante scrivere il proprio main in Visual Studio per poter fare correttamente il debug delle funzioni realizzate!

Esercizio 1 (6 punti)

Creare i file `matrix.h` e `matrix.c` che consentano di utilizzare la seguente struttura:

```
struct matrix {
	size_t rows, cols;
	double *data;
};
```

e la funzione:

```
extern struct matrix *mat_transpose(const struct matrix *mat);
```

La struct consente di rappresentare matrici di dimensioni arbitraria, dove `rows` è il numero di righe, `cols` è il numero di colonne e `data` è un puntatore a `rows×cols` valori di tipo `double` memorizzati per righe. Consideriamo ad esempio la matrice

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

questo corrisponderebbe ad una variabile `struct matrix A`, con `A.rows = 2`, `A.cols = 3` e `A.data` che punta ad un area di memoria contenente i valori `{1.0, 2.0, 3.0, 4.0, 5.0, 6.0 }`.

Si dice *matrice trasposta* di $A = \left(a_j^i\right) \in \mathcal{M}_{m\times n}(X)$ la matrice $A^T = \left(b_k^h\right) \in \mathcal{M}_{n\times m}(X)$ i cui elementi, per ogni $h \in \mathbb{N}_n$ e $k \in \mathbb{N}_m$ sono definiti come segue:

$$b_k^h = a_h^k.$$

La matrice A^T si ottiene dunque semplicemente considerando come colonne le righe di A e viceversa. La trasposta della matrice precedente è

$$A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

La funzione accetta come parametro un puntatore ad una matrice e deve ritornarne la trasposta, allocata dinamicamente sull'heap. Se il puntatore passato alla funzione è `NULL`, la funzione ritorna `NULL`.

Esercizio 2 (5 punti)

Nel file `crescenti.c` implementare la definizione della funzione:

```
extern bool crescente(unsigned int x);
```

La funzione prende come input il valore `x` e ritorna `true` se il numero è crescente, `false` altrimenti.

Un numero viene detto "crescente", se ogni cifra della sua rappresentazione in base 10 è seguita dalla cifra che ha valore successivo o è l'ultima del numero.

Ad esempio:
123 → 1 è seguito da 2 (ok), 2 è seguito da 3 (ok), 3 è l'ultima cifra (ok) → è crescente
5 → 5 è l'ultima cifra (ok) → è crescente
124 → 1 è seguito da 2 (ok), 2 è seguito da 4 (errore) → non è crescente

Esercizio 3 (7 punti)

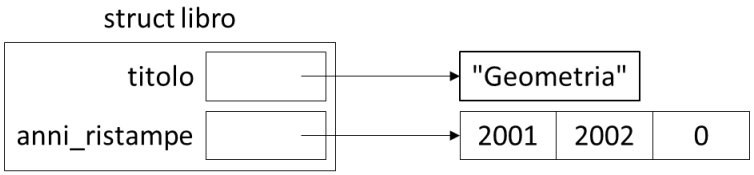
Creare i file `libri.h` e `libri.c` che consentano di utilizzare la seguente struttura:

```
struct libro {
    char *titolo;
    uint16_t *anni_ristampe;
};
```

e la funzione:

```
extern bool libro_scrivi(const struct libro *p, FILE *f);
```

La funzione riceve un puntatore ad un elemento di tipo `struct libro` e deve scriverne i dati sul file già aperto in modalità scrittura non tradotta (binaria) passato come parametro. Il `titolo` viene scritto come sequenza di byte terminata con un ulteriore byte uguale a 0 come le stringhe C, il campo `anni_ristampe` punta al primo di una sequenza di interi senza segno a 16 bit terminata con uno di questi che vale 0. Deve essere scritta su file in little endian (esattamente come in memoria nei processori Intel). La funzione ritorna true se tutto va bene, o false se la scrittura fallisce.
Il seguente libro:



Verrebbe scritto su file come:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  47 65 6F 6D 65 74 72 69 61 00 D1 07 D2 07 00 00  Geometria.Ñ.Ò...
```

Esercizio 4 (7 punti)

Creare i file `matrix.h` e `matrix.c` che consentano di utilizzare la seguente struttura:

```
struct matrix {
	size_t rows, cols;
	double *data;
};
```

e la funzione:

```
extern struct matrix *scambia_diagonali(const struct matrix *m);
```

La struct consente di rappresentare matrici di dimensioni arbitraria, dove `rows` è il numero di righe, `cols` è il numero di colonne e `data` è un puntatore a `rows×cols` valori di tipo `double` memorizzati per righe. Consideriamo ad esempio la matrice

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

questo corrisponderebbe ad una variabile `struct matrix A`, con `A.rows = 2`, `A.cols = 3` e `A.data` che punta ad un area di memoria contenente i valori `{1.0, 2.0, 3.0, 4.0, 5.0, 6.0 }`.

La funzione accetta come parametri un puntatore ad una matrice quadrata `m` e deve restituire un puntatore a una nuova matrice allocata dinamicamente che contenga la matrice ottenuta scambiando la diagonale principale con l'antidiagonale, ovvero la diagonale che va dall'angolo in alto a destra all'angolo in basso a sinistra.
Ad esempio la matrice

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

produce la matrice

$$\begin{pmatrix} 3 & 2 & 1 \\ 4 & 5 & 6 \\ 9 & 8 & 7 \end{pmatrix}$$

Se il puntatore passato alla funzione è `NULL` o se la matrice non è quadrata, la funzione ritorna `NULL`.