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 (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 2 (6 punti)

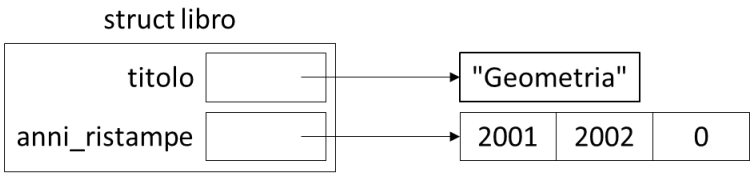
Creare i file `libri.h` e `libri.c` che consentano di utilizzare la seguente struttura:

```
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 3 (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`.

Esercizio 4 (7 punti)

Creare i file `ip.h` e `ip.c` che consentano di utilizzare la seguente funzione:

```
extern uint32_t *leggi_indirizzi_ip(const char *filename, size_t *size);
```

La funzione riceve il nome di un file da aprire in modalità lettura tradotta (testo), e un puntatore ad un size_t. Il file contiene una sequenza di righe contenenti indirizzi IP in formato testo, ovvero quattro numeri interi da 0 a 255 separati da tre punti. Ad esempio:

```
127.0.0.1↵
10.0.42.3↵
78.15.46.23↵
```

La funzione deve convertire il file in una sequenza allocata dinamicamente di numeri interi a 32 bit in big endian. Ogni numero è formato dai 4 byte corrispondenti ai valori degli indirizzi IP. Nel caso precedente, la funzione dovrebbe allocare 3 interi a 32 bit, che contengano i valori 0x7F000001, 0x0A002A03, 0x4E0F2E17. La variabile puntata da `size` deve essere impostata al numero di valori letti.

Se il file è vuoto o non esiste, se contiene numeri non compresi nell'intervallo rappresentabile in un byte o se contiene linee incomplete, la funzione ritorna NULL e imposta size a 0.

Esercizio 5 (8 punti)

Creare i file `string_split.h` e `string_split.c` che consentano di utilizzare la seguente funzione:

```
extern void string_split(const char *str, size_t index, char **s1, char **s2);
```

La funzione prende come parametri una stringa C, un `size_t` e due puntatori a puntatore a `char`. La funzione deve dividere la stringa in due in corrispondenza dell'indice passato nella variabile `index` e modificare i puntatori puntati da `s1` e `s2` in modo che puntino a due nuove stringhe allocate dinamicamente contenenti le due parti della stringa.

Consideriamo ad esempio la stringa "alfabetizzazione" da spezzare all'indice 6. In questo caso si otterranno due stringhe: la prima contenente le prime 6 lettere ("alfabe") e la seconda contenente le restanti 10 ("tizzazione").

Se la stringa è `NULL` o se l'indice è maggiore del numero di caratteri della stringa, la funzione imposta i puntatori puntati da `s1` e `s2` a `NULL`.