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)

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

```
extern bool palindroma(const char *str);
```

La funzione accetta come parametro un stringa C e verifica che questa sia palindroma, ovvero che sia uguale leggerla da sinistra verso destra o da destra verso sinistra. Ad esempio, la stringa "albero" non è palindroma e la funzione ritorna `false`, mentre "al bb la" è palindroma e la funzione ritorna `true`. Se `str` è `NULL`, oppure punta ad una stringa vuota, la funzione ritorna `false`. Una stringa con un solo carattere è palindroma.

Esercizio 2 (6 punti)

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

```
extern void stampa_croce(FILE *f, size_t dim);
```

La funzione deve scrivere sul file f (fornito già aperto in modalità tradotta/testuale) una X composta dai caratteri '\' e '/' sulle diagonali. Ogni semi-diagonale deve essere composta di `dim` caratteri. Ad esempio chiamando la funzione con `dim=0`, la funzione non deve scrivere nulla sul file. Chiamando la funzione con `dim=1`, la funzione deve scrivere sul file:

```
\/↵
/\↵
```

Chiamando la funzione con `dim=2`, la funzione deve scrivere sul file:

```
\  /↵
 \/↵
 /\↵
/  \↵
```

Ovvero (visualizzando ogni carattere in una cella della seguente tabella):

\	<spazio>	<spazio>	/	<a capo>
<spazio>	\	/	<a capo>	
<spazio>	/	\	<a capo>	
/	<spazio>	<spazio>	\	<a capo>

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 *matrix_flip_v(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 unico parametro un puntatore a una `struct matrix` e deve ritornare un puntatore a una nuova `struct matrix` allocata dinamicamente sull'heap avente le stesse dimensioni di `m` e il contenuto di `m` ribaltato verticalmente. Ad esempio, la seguente matrice M di dimensioni 3x3:

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

deve essere ribaltata verticalmente producendo la seguente matrice:

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

Se il puntatore `m` passato come parametro alla funzione è `NULL` la funzione non fa nulla e ritorna `NULL`.

Esercizio 4 (7 punti)

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

```
struct rational {
	int num;
	unsigned int den;
};
```

e la funzione:

```
extern struct rational *rational_read(const char *filename, size_t *size);
```

La funzione riceve in input un nome di file `filename` che deve aprire in modalità lettura tradotta (testo). Se il file esiste, legge dal file sequenze di caratteri che descrivono delle frazioni. Il formato di ogni frazione è il seguente:

- zero o più spazi, tab o a capo
- intero con o senza segno
- zero o più spazi, tab o a capo
- il carattere /
- zero o più spazi, tab o a capo
- intero senza segno

Ad esempio il seguente è un file valido:

```
3/2 -1 /  4 ↵
2↵
/   ↵
2↵
```

e corrisponde al vettore di frazioni $\left[\frac{3}{2}, -\frac{1}{4}, \frac{2}{2}\right]$.

La funzione deve allocare dinamicamente sull'heap spazio sufficiente a contenere tutte le frazioni leggibili correttamente dal file, riempirlo con i valori opportuni, impostare la variabile puntata da `size` a questo numero e infine ritornare un puntatore all'area così allocata.

Se è impossibile aprire il file o questo non contiene nulla, la funzione ritorna `NULL` e imposta la variabile puntata da `size` a 0. Se durante la lettura si verifica un errore, il vettore risultante conterrà solo i valori letti fino a quel punto.

Esercizio 5 (8 punti)

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

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

La funzione riceve in input un nome di file `filename` che deve aprire in modalità lettura non tradotta (binaria). Se il file esiste, legge dal file una serie di stringhe.

Il formato del file è il seguente:

- Un intero `N` senza segno in little endian a 32 bit che indica il numero totale di stringhe presenti nel file.
- Per ognuna delle `N` stringhe:
  - Un intero `dim` senza segno in little endian a 32 bit che indica il numero totale di caratteri della stringa.
  - I `dim` caratteri, ognuno da un byte.

Ad esempio (ogni cella della tabella rappresenta un byte del file):

02	00	00	00	04	00	00	00	c	i	a	o	06	00	00	00	a	l	b	e	r	o
Num. di stringhe				Dim. della prima				caratteri…				Dim. della seconda				caratteri…					

Questo file contiene 2 stringhe, la prima da 4 caratteri (“`ciao`”) e la seconda da 6 caratteri (“`albero`”).

La funzione deve allocare dinamicamente sull’heap e ritornare un vettore di puntatori a char, ognuno dei quali è a sua volta allocato e punta all’area di memoria in cui è memorizzata secondo lo standard del linguaggio C una delle stringhe lette. Infine deve impostare la variabile puntata da `size` alla dimensione del vettore.

Se durante la lettura si verifica un errore, ad esempio se riesce a leggere meno di `N` stringhe o se una delle stringhe contiene meno di `dim` caratteri, la funzione deve impostare la variabile puntata da `size` a 0 e ritornare `NULL`.