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)

In statistica, la deviazione standard è un indice statistico che fornisce informazioni sulla variabilità di un insieme di dati. La formula per calcolarla è la seguente:

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{N}(x_i - m)^2}{N}}$$

Dove N è il numero di elemento presenti nell'insieme di dati e m è la media dei valori dell'insieme di dati.

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

```
extern double calcola_dev(const double *vec, size_t size);
```

La funzione prende come input un vettore di double, `vec` è un puntatore al primo elemento e `size` è il numero di elementi prensenti nel vettore, e ne calcola la deviazione standard ritornandola in un double.

I parametri passati alla funzione saranno sempre validi, `N>0`, non riceverà mai puntatori `NULL`.

Esercizio 2 (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 *matrix_flip_h(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 orizzontalmente. 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:

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

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

Esercizio 3 (7 punti)

Nel gioco Risiko, quando un giocatore ne attacca un altro, si decide il numero di armate perse da entrambe le parti lanciando dei dadi. Attaccante e difensore lanciano fino a un massimo di tre dadi e i risultati vengono confrontati in ordine di valore, il più alto dell'attacco con il più alto della difesa, il secondo dell'attacco con il secondo della difesa e il più basso dell'attacco con il più basso della difesa. Se uno dei due giocatori ha lanciato meno di tre dadi il confronto si ferma prima.
Uno dei due giocatori perde un'armata ogni volta che perde un confronto tra dadi, in caso di pareggio vince la difesa.
Ecco alcuni esempi:

Attacco	5	4	1
Difesa	6	3	1

In questo caso l'attacco perde due armate e la difesa una sola.

Attacco	6	3	
Difesa	5	3	1

In questo invece sia l'attacco che la difesa ne perdono una.

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

```
struct lancio {
	char valori[3];
	char n_dadi;
};
```

e la seguente funzione:

```
extern void confronta_lanci_ord(const struct lancio *attacco, const struct lancio *difesa,
	char *perse_attacco, char *perse_difesa);
```

La funzione deve determinare il numero di armate perse da entrambi i giocatori.
La struttura `lancio` contiene un array di tre interi senza segno che contengono i valori dei dadi lanciati e un ulteriore intero senza segno che indica quanti dadi sono stati effettivamente lanciati.
La funzione deve impostare la variabile puntata da `perse_attacco` con il numero di armate perse dall'attaccante e la variabile puntata da `perse_difesa` con il numero di armate perse dal difensore.

Prestare particolare attenzione al fatto che gli array con i valori dei dadi vengono passati alla funzione **non** ordinati. Tutti i puntatori passati alla funzione non saranno mai `NULL`.

Esercizio 4 (7 punti)

Il linguaggio HTML è un linguaggio di formattazione (di *markup*) che descrive le modalità di impaginazione o visualizzazione grafica (layout) del contenuto, testuale e non, di una pagina web. L'HTML descrive il layout e la formattazione di una pagina web attraverso l'inserimento nel testo di marcatori, chiamati *tag*, che descrivono caratteristiche come il colore, le dimensioni, la posizione relativa all'interno della pagina. I browser interpretano il codice HTML mostrando la pagina all'utente rispettandone la formattazione. Ogni tag specifica un diverso ruolo dei contenuti che esso contrassegna. Tutti i tag HTML sono sempre racchiusi tra parentesi angolari (un minore e un maggiore). Di seguito è riportato come esempio un semplicissimo file HTML:

```
<!DOCTYPE html>↲
<html>↲
····<body>↲
········<h1>Un titolo</h1>↲
········<p>Il primo paragrafo.</p>↲
········<p>Il secondo paragrafo.</p>↲
········<p>Il terzo paragrafo.</p>↲
····</body>↲
</html>↲
```

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

```
extern int rimuovi_html(const char *filein, const char *fileout);
```

La funzione prende come input due stringhe C che rappresentano i nomi di due file, il primo è il nome di un file HTML da aprire in modalità lettura tradotta (testo) mentre il secondo è il nome di un file di testo da aprire in modalità scrittura tradotta (testo). La funzione deve leggere il contenuto del primo file, rimuovere tutti i tag HTML, ossia tutto ciò che è compreso tra '<' e '>', comprese le parente angolari stesse, e scrivere il contenuto del file senza tag sul secondo file aperto in scrittura. Ad esempio, considerando come file di input quello riportato in precedenza, sul file di output deve essere presente il seguente contenuto:

```
↲
↲
····↲
········Un titolo↲
········Il primo paragrafo.↲
········Il secondo paragrafo.↲
········Il terzo paragrafo.↲
····↲
↲
```

La funzione deve ritornare 0 in caso di successo oppure -1 se si verifica un errore, ad esempio se non è possibile aprire uno dei due file o se il file non rispetta la sintassi HTML (ossia per un tag

viene individuata la parentesi angolare di apertura '<' ma non quella di chiusura '>'). Le stringhe passate come parametro non saranno mai `NULL`.

Esercizio 5 (8 punti)

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

```
extern void rimuovi_stringa(char *parola, const char *str);
```

La funzione accetta in input due stringhe C e deve rimuovere dalla stringa `parola` tutte le occorrenze della stringa `str`, ad esempio, se `parola` è la stringa "le lezioni di informatica sono finite" e `str` è "in", la funzione deve modificare `parola` rimuovendo tutte le occorrenze di "in", ottenendo la stringa "le lezioni di formatica sono fite".

La stringa `parola` non sarà mai `NULL` e nel caso che la stringa `str` sia `NULL` o vuota la funzione deve terminare senza modificare `parola`.