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 `punto.h` e `punto.c` che consentano di utilizzare le seguenti strutture:

```
struct punto_cart {
	double x, y;
};

struct punto_pol {
	double r, t;
};
```

e le seguenti funzioni:

```
extern struct punto_pol cartesiano2polare(const struct punto_cart *p);

extern struct punto_cart polare2cartesiano(const struct punto_pol *p);
```

La prima funzione accetta un puntatore a un punto in coordinate cartesiane e deve restituire una struct `punto_pol` contenente il punto `p` convertito in coordinate polari. Le formule per la conversione sono le seguenti:

$$r = \sqrt{x^2 + y^2}$$
$$t = \arctan2(y, x)$$

La seconda funzione accetta un puntatore a un punto in coordinate polari e deve restituire una struct `punto_cart` contenente il punto `p` convertito in coordinate cartesiane. Le formule per la conversione sono le seguenti:

$$x = r * \cos(t)$$
$$y = r * \sin(t)$$

Gli angoli sono sempre espressi in radianti. Il punto passato alle funzioni non saranno mai `NULL`. La funzione $arctan2()$ è disponibile nella libreria `math.h` con il nome di `atan2()`. Attenzione all'ordine di `x` e `y` da passare a `atan2()`!

Esercizio 2 (6 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(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.

Gli array con i valori dei dadi vengono passati già ordinati in ordine discendente, ossia `valori[0]` conterrà il valore del dado più alto, `valori[1]` quello del secondo e `valori[2]` quello del più basso.
Tutti i puntatori passati alla funzione non saranno mai `NULL`.

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_quadruplica(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 che contiene una versione quadruplicata della matrice `m` passata come parametro.
Con matrice quadruplicata si intende una matrice che come dimensioni ha le dimensioni di quella originale raddoppiate e come contenuto il contenuto di quella originale copiato quattro volte negli angoli di quella nuova. Ad esempio, se alla funzione viene passata seguente matrice M di dimensioni 2x2:

$$M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Deve essere quadruplicata producendo la seguente matrice Mq di dimensioni 4x4:

$$Mq = \begin{pmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{pmatrix}$$

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 `byteswap.h` e `byteswap.c` che consentano di utilizzare la seguente funzione:

```
extern uint32_t byteswap(uint32_t n);
```

La funzione riceve in input un intero senza segno a 32 bit in little endian e deve restituire, sempre come intero senza segno a 32 bit, la sua rappresentazione in big endian. Ad esempio, dato il numero intero espresso in esadecimale `0x0a0b0c0d`, la funzione deve restituire il numero `0x0d0c0b0a`

Esercizio 5 (8 punti)

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

```
struct persona {
	int anni;
	char nome[50];
};
```

e la seguente funzione:

```
extern struct persona *leggi_persone(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 persone e la loro età.
Nel file è indicato, come primo valore, un numero intero che rappresenta il numero totale di persone elencate nel file seguito da un a capo. Successivamente, su ogni riga sono presenti i dati delle persone nel seguente formato:

- Zero o più spazi o tab.
- Un numero intero che indica l'età della persona.
- Zero o più spazi o tab.
- Il carattere virgola
- Zero o più spazi o tab.
- Una stringa che può contenere spazi e tab che rappresenta il nome della persona.
- Un a capo.

Quello che segue è un esempio corretto di file:

```
3↵
18 , Marco Rossi↵
  22  ,   Luca Verdi↵
      25,          Marcello Bianchi↵
```

La funzione deve allocare dinamicamente sull'heap spazio sufficiente a contenere tutte le persone 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, questo non contiene nulla o si verifica un errore durante la lettura, ad esempio nel file ci sono meno persone di quelle indicate dal primo numero intero, la funzione ritorna `NULL` e imposta la variabile puntata da `size` a 0.