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)

La tavola pitagorica è una matrice di numeri caratterizzata dal fatto che il valore alla colonna j-esima della riga i-esima è il prodotto di i per j. Arrivando fino a 10 la tavola è la seguente:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

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

```
extern void stampa_tavola_pitagorica(FILE *f, unsigned int n);
```

La funzione prende come input un puntatore a FILE già aperto in modalità tradotta (testo) e un intero senza segno n e deve scrivere sul file i valori della tavola pitagorica arrivando fino a n, separati da tabulazioni. Ogni riga deve terminare con un carattere a capo. Ad esempio con n = 5 si otterrebbe in output:

```
1	2	3	4	5↵
2	4	6	8	10↵
3	6	9	12	15↵
4	8	12	16	20↵
5	10	15	20	25↵
```

Esercizio 2 (6 punti)

Il triangolo di Tartaglia è una disposizione geometrica dei coefficienti binomiali a forma di triangolo. Le prime righe del triangolo di Tartaglia sono le seguenti:

```
                    1     n=0
                 1     1     n=1
              1     2     1     n=2
           1     3     3     1     n=3
        1     4     6     4     1     n=4
     1     5     10    10    5     1     n=5
  1     6     15    20    15    6     1     n=6
```

In ogni riga, il primo e l'ultimo elemento sono 1, mentre ogni altro elemento si trova come somma degli elementi della riga precedente nella stessa posizione e in quella precedente. Convenzionalmente si indica la prima riga come riga 0 e la si fa contenere un solo 1.

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

```
extern unsigned int *tartaglia(unsigned int *v, size_t n);
```

La funzione prende come input un vettore di numeri di dimensione n, che contiene la (n-1)-esima riga del triangolo di Tartaglia e deve allocare dinamicamente su heap lo spazio per n+1 numeri in cui inserire la riga di posizione n. Se n è 0 ritorna un vettore con un solo 1 (non deve accedere a v), se n è 1 ritorna un vettore con due 1 (non deve accedere a v), in tutti gli altri casi utilizza il vettore corrispondente alla riga precedente, fornito in input.

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 *mat_permute_rows(const struct matrix *m, const size_t *p);
```

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 `m` e un puntatore ad un vettore di indici `p` e deve restituire un puntatore a una nuova matrice allocata dinamicamente. La matrice è ottenuta mettendo nella riga i-esima la riga della matrice originale il cui indice è specificato alla posizione i-esima del vettore p.

Ad esempio, data la matrice

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

e il vettore di indici

$$\begin{pmatrix} 2 & 1 & 0 \end{pmatrix}$$

la funzione restituisce la matrice

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

ovvero la matrice ottenuta mettendo una sotto l'altra le righe 2, 1 e 0 della matrice originale.
I puntatori e tutti i dati di input saranno sempre validi, quindi non serve fare alcun controllo.

Esercizio 4 (7 punti)

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

```
struct seq {
	size_t len;
	uint16_t *values;
};
```

e la funzione:

```
extern bool seq_load(const char *filename, struct seq *s);
```

La struct consente di rappresentare sequenze di interi senza segno a 16 bit di lunghezza arbitraria, dove `len` contiene il numero di elementi e `values` punta ad un area di memoria di dimensione opportuna.

La funzione accetta come parametri una stringa C con il nome di un file e un puntatore ad una struct seq in cui inserire i dati letti da file. Il formato del file è semplicemente una sequenza di interi a 16 bit senza segno memorizzati in little endian (come nei processori Intel) senza separatori di alcun tipo. La funzione deve aprire il file in modalità non tradotta (binaria), leggere i dati del file allocando dinamicamente memoria per contenere i valori e impostare opportunamente i campi della struct il cui indirizzo è stato passato alla funzione. La funzione ritorna true se è possibile aprire il file, false altrimenti.

Ad esempio, una sequenza di tre valori 1, 2, 3, verrebbe scritta su file come:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  01 00 02 00 03 00                                      ......
```

La funzione ritornerebbe true e metterebbe 3 nel campo len e un puntatore ai 6 byte letti in values.

Esercizio 5 (8 punti)

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

```
extern char *center(const char *str, size_t n, char c);
```

La funzione riceve in input un puntatore a una stringa C `str`, una larghezza `n` e un carattere di riempimento `c`. In uscita si deve produrre una nuova stringa C allocata dinamicamente che contenga la stringa `str` centrata su una larghezza di `n` caratteri. Per riempire a sinistra e a destra si deve utilizzare il carattere `c`. Ad esempio centrare la stringa "ciao" su una larghezza `n` di 10 caratteri utilzzando il carattere "." come riempimento produce: `...ciao...`
Quando però:

- la stringa è di lunghezza dispari e `n` è pari, il carattere di riempimento in più viene messo a destra;
- la stringa è di lunghezza pari e `n` è dispari, il carattere di riempimento in più viene messo a sinistra;
- la lunghezza della stringa è maggiore o uguale a n, non si inserisce alcun riempimento;
- il puntatore è NULL, la funzione ritorna NULL.

Ecco un esempio per tutti i casi possibili:

str	**n**	**output**
ciao	10	...ciao...
ciao!	10	..ciao!...
ciao	11	....ciao...
ciao!	11	...ciao!...
ciao!	2	ciao!
NULL	5	NULL