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

```
extern int trova_diverso_dalla_fine(const char *str, char c);
```

La funzione deve trovare nella stringa `str` l'indice della prima occorrenza di un carattere diverso da `c` a partire dalla fine.
Ad esempio, se la funzione viene chiamata con `str = "Analisi Matematica II"` e `c = 'I'` dovrà ritornare l'indice del primo carattere diverso da `'I'` a partire dalla fine della stringa, in questo caso dovrà ritornare l'indice dello spazio che c'è tra "Matematica" e "II", ossia 18.

Se il puntatore alla stringa è `NULL` o se non viene trovato nessun carattere diverso da `c` allora la funzione deve ritornare -1.

Esercizio 2 (6 punti)

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

```
extern void stampa_cornicetta(const char *s);
```

La funzione deve inviare a `stdout` la stringa passata come parametro circondandola con una cornicetta con un doppio bordo. Il bordo più esterno è composto dai caratteri + e = per le righe orizzontali e il carattere | per le righe verticali. Il bordo più interno è composto dai caratteri * e – (meno) per le righe orizzontali e dal carattere | per le righe verticali. Lungo le righe verticali dei due bordi sono separate da uno spazio. Anche la stringa viene separata da uno spazio (prima e dopo) dalle righe verticali del bordo interno. Ogni riga, anche l'ultima, termina con un "a capo".
Ad esempio chiamando la funzione con `s = "ciao"`, la funzione deve inviare su `stdout`:

```
+==========+↲
| *------* |↲
| | ciao | |↲
| *------* |↲
+==========+↲
```

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

+	=	=	=	=	=	=	=	=	=	=	+	<a capo>
|	<sp>	*	-	-	-	-	-	-	*	<sp>	|	<a capo>
|	<sp>	|	<sp>	c	i	a	o	<sp>	|	<sp>	|	<a capo>
|	<sp>	*	-	-	-	-	-	-	*	<sp>	|	<a capo>
+	=	=	=	=	=	=	=	=	=	=	+	<a capo>

Esercizio 3 (7 punti)

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

```
extern int felice(unsigned int num);
```

La funzione prende come input il valore `num` e ritorna 1 se il numero è felice, 0 se è infelice.

Un numero felice è definito tramite il seguente processo: partendo con un qualsiasi numero intero positivo, si sostituisca il numero con la somma dei quadrati delle sue cifre, e si ripeta il processo fino a quando si ottiene 1 (dove ulteriori iterazioni porteranno sempre 1), oppure si entra in un ciclo che non include mai 1. I numeri per cui tale processo dà 1 sono numeri felici, mentre quelli che non danno mai 1 sono numeri infelici. È possibile dimostrare che se nella sequenza si raggiunge il 4, il numero è infelice. Possiamo estendere il concetto allo 0, che ovviamente genera la sequenza composta solo di 0 e quindi possiamo considerarlo infelice.

Ad esempio, 7 è felice e la sequenza ad esso associata è:

$$7 \to 7^2 = 49 \to 4^2 + 9^2 = 97 \to 9^2 + 7^2 = 130 \to 1^2 + 3^2 + 0^2 = 10 \to 1^2 + 0^2 = 1$$

mentre 8 è infelice e la sequenza ad esso associata è:

$$8 \to 8^2 = 64 \to 6^2 + 4^2 = 52 \to 5^2 + 2^2 = 29 \to 2^2 + 9^2 = 85 \to 8^2 + 5^2 = 89 \to$$
$$8^2 + 9^2 = 145 \to 1^2 + 4^2 + 5^2 = 42 \to 4^2 + 2^2 = 20 \to 2^2 + 0^2 = 4$$

Esercizio 4 (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 *prod_kronecker(const struct matrix *a, const struct matrix *b);
```

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 due puntatori a due matrici `a` e `b` e deve restituire un puntatore a una nuova matrice allocata dinamicamente che contiene il prodotto di Kronecker tra `a` e `b`.
Il prodotto di Kronecker tra due matrici è sempre applicabile. Se A è una matrice $m \times n$ e B è una matrice $p \times q$ il loro prodotto di Kronecker è una matrice $mp \times nq$ definita a blocchi come segue:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

La matrice risultante è una matrice a blocchi composta dalla ripetizione della matrice B, moltiplicata ogni volta per un elemento di A, in una griglia di $m \times n$ ripetizioni.
Ad esempio, considerando le matrici A e B così definite:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix}$$

Il prodotto di Kronecker deve essere calcolato come segue:

$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1\cdot 0 & 1\cdot 3 & 2\cdot 0 & 2\cdot 3 \\ 1\cdot 2 & 1\cdot 1 & 2\cdot 2 & 2\cdot 1 \\ 3\cdot 0 & 3\cdot 3 & 1\cdot 0 & 1\cdot 3 \\ 3\cdot 2 & 3\cdot 1 & 1\cdot 2 & 1\cdot 1 \\ 2\cdot 0 & 2\cdot 3 & 4\cdot 0 & 4\cdot 3 \\ 2\cdot 2 & 2\cdot 1 & 4\cdot 2 & 4\cdot 1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 6 \\ 2 & 1 & 4 & 2 \\ 0 & 9 & 0 & 3 \\ 6 & 3 & 2 & 1 \\ 0 & 6 & 0 & 12 \\ 4 & 2 & 8 & 4 \end{bmatrix}$$

Se uno dei due puntatori passati alla funzione è `NULL` la funzione ritorna `NULL`.

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 accetta in input una stringa C che contiene il nome di un file da aprire in modalità lettura non tradotta (binaria) e un puntatore a una variabile di tipo `size_t`. La funzione deve ritornare un vettore allocato dinamicamente di stringhe C (anch'esse allocate dinamicamente) contenente tutte le stringhe lette dal file e aggiornare la variabile puntata da `size` con il numero di stringhe lette. Il vettore verrà liberato dal chiamante liberando con la `free()` prima ogni stringa e successivamente il puntatore stesso.

Le stringhe sono memorizzate sul file nel seguente modo: per ogni stringa vengono scritti sul file i byte che la compongono e alla fine viene scritto sul file un byte dal valore zero per indicarne la fine. Si potrebbe dire che le stringhe sono zero-terminate anche sul file. Tutte le stringhe sono zero terminate compresa l'ultima, di conseguenza se l'ultimo byte del file non è uno zero allora si è verificato un errore.

Ad esempio, le seguenti tre stringhe: "Analisi Matematica I", "Fondamenti di Informatica I" e "Geometria" verranno memorizzate su file come segue:

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

00000000  41 6E 61 6C 69 73 69 20 4D 61 74 65 6D 61 74 69  Analisi Matemati
00000010  63 61 20 49 00 46 6F 6E 64 61 6D 65 6E 74 69 20  ca I.Fondamenti
00000020  64 69 20 49 6E 66 6F 72 6D 61 74 69 63 61 20 49  di Informatica I
00000030  00 47 65 6F 6D 65 74 72 69 61 00                 .Geometria.
```

Se il file non può essere aperto, se si verifica un errore durante la lettura o se non è possibile leggere nessuna stringa dal file allora la funzione deve ritornare `NULL` e impostare il valore puntato da `size` a 0.