LEGGETE LA GUIDA PER LA CREAZIONE DEI PROGETTI E PER IL DEBUGGING!

Gli esercizi seguenti devono essere risolti, compilati e testati utilizzando il debugger. Per ognuno si deve realizzare una funzione main() che ne testi il funzionamento. ***Fate progetti diversi per ogni esercizio.***

Attenzione! La Microsoft ha definito funzioni non standard più sicure di quelle definite nello standard e segnala l'uso di funzioni considerate pericolose con questo warning/errore:

```
warning C4996: '<nome funzione>': This function or variable may be unsafe. Consider using
<nome funzione sicura> instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online
help for details.
```

In questo corso utilizzeremo solo le versioni standard e quindi per convincere Visual Studio a fare il suo dovere aggiungete **come prima riga di ogni file .c che produce l'errore** la seguente definizione

```
#define _CRT_SECURE_NO_WARNINGS
```

Esercizio 1

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_transpose(const struct matrix *mat);
```

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 }`.

Si dice *matrice trasposta* di $A = \left(a_j^i\right) \in \mathcal{M}_{m\times n}(X)$ la matrice $A^T = \left(b_k^h\right) \in \mathcal{M}_{n\times m}(X)$ i cui elementi, per ogni $h \in \mathbb{N}_n$ e $k \in \mathbb{N}_m$ sono definiti come segue:

$$b_k^h = a_h^k.$$

La matrice A^T si ottiene dunque semplicemente considerando come colonne le righe di A e viceversa. La trasposta della matrice precedente è

$$A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

La funzione accetta come parametro un puntatore ad una matrice e deve ritornarne la trasposta, allocata dinamicamente sull'heap. Il puntatore alla matrice non sarà mai `NULL`.

Esercizio 2

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 composta dei caratteri ‘\’ e ‘/’ agli spigoli e di ‘-’ e ‘|’ sui lati. Prima e dopo la stringa bisogna inserire uno spazio. 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.>	c	i	a	o	<sp.>	|	<a capo>
\	-	-	-	-	-	-	/	<a capo>

Si ricorda che in C il carattere ‘\’ deve essere inserito come ‘\\’. Gli <a capo> a fine riga sono obbligatori per una soluzione corretta.

Esercizio 3

Creare il file `trim.c` che consenta di utilizzare la seguente funzione:

```
extern char *trim(const char *s);
```

La funzione accetta una stringa zero terminata e ritorna un’altra stringa zero terminata, allocata dinamicamente nell’heap, contenente i caratteri della stringa in ingresso, senza tutti gli spazi iniziali e finali. La funzione deve ritornare `NULL` (e quindi non allocare memoria) se `s` è `NULL`.
Ad esempio:

```
“senza spazi”     → “senza spazi”
“ prima”          → “prima”
“dopo ”           → “dopo”
“ a b ”           → “a b”
“ ”               → “”
```

Esercizio 4

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:

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