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 matematica, un'equazione diofantea lineare in due incognite è una equazione con coefficienti interi, ovvero $ax + by = c$, con $a, b, c \in \mathbb{Z}$ e a e b non entrambi nulli, di cui si ricercano le soluzioni intere, cioè coppie $(\bar{x}, \bar{y}) \in \mathbb{Z} \times \mathbb{Z}$. È possibile dare un semplice criterio di risolubilità: l'equazione diofantea ha soluzioni intere se e solo se c è divisibile per il massimo comun divisore di a e b.

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

```
extern int ammette_soluzioni(int a, int b, int c);
```

La funzione prende come input i tre parametri a, b e c dell'equazione diofantea $ax + by = c$ e ritorna 1 se l'equazione ammette soluzioni, 0 altrimenti. Nel caso in cui a e b siano entrambi nulli, la funzione ritorna 1 se $c = 0$, 0 altrimenti.

Ricordo che il massimo comune divisore di due numeri interi a e b, che non siano entrambi uguali a zero, è il più grande numero naturale di cui sia a sia b sono multipli. Per questa definizione, $\mathrm{MCD}(a, b) = \mathrm{MCD}(|a|, |b|)$.

Esercizio 2 (6 punti)

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

```
extern void stampa_cornice(size_t n);
```

La funzione deve inviare a `stdout` tre righe contenenti il seguente disegno:

```
 .--.
:::::.\::::
      `--'
```

ripetuto orizzontalmente `n` volte, con `n>0`. Ogni riga deve essere terminata con un carattere <a capo>. È possibile osservare che i caratteri utilizzati sono punto `(.)`, meno `(-)`, due punti `(:)`, barra diagonale rovesciata `(\)` e due tipi di apici, quello classico `(')` disponibile sulla tastiera italiana e quello rovesciato `(`)`, che ha codice ASCII 96.

Se chiamiamo la funzione con `n=5` otteniamo su `stdout` le tre righe seguenti:

```
 .--.      .--.      .--.      .--.      .--.
:::::.\::::::::.\::::::::.\::::::::.\::::::::.\::::
      `--'      `--'      `--'      `--'      `--'
```

Esercizio 3 (7 punti)

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

```
#include <stdint.h> // Per avere uint8_t
struct image {
	size_t rows, cols;
	uint8_t *data;
};
```

e la funzione:

```
extern struct image *aggiungi_cornice(const struct image *img);
```

La struct consente di rappresentare immagini a livelli di grigio 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 `uint8_t` memorizzati per righe. Fondamentalmente un'immagine è una matrice di numeri dove ogni numero indica quanto è luminoso il pixel. Consideriamo ad esempio l'immagine larga 3 pixel e alta 2:

1 2 3
4 5 6

questo corrisponderebbe ad una variabile `struct image img`, con `img.rows = 2`, `img.cols = 3` e `img.data` che punta ad un area di memoria contenente i valori `{ 1, 2, 3, 4, 5, 6 }`.
La funzione accetta come parametro un puntatore ad una immagine e deve ritornare una immagine, allocata dinamicamente sull'heap, con una cornice di zeri attorno.
Ad esempio l'immagine 2x2 seguente:

1 2
3 4

diventerebbe:

0 0 0 0
0 1 2 0
0 3 4 0
0 0 0 0

Se il puntatore in input è NULL, la funzione ritorna NULL.

Esercizio 4 (7 punti)

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

```
extern int *unici(const int *vec, size_t size, size_t *newsize);
```

La funzione accetta in input un puntatore ad un vettore di `size` numeri interi e deve ritornare un nuovo vettore allocato dinamicamente sull'heap che contenga i valori di `vec` senza ripetizioni, ovvero contenga solo valori unici. La funzione imposta anche il parametro `newsize` al numero di elementi così trovato.
Ad esempio se passiamo alla funzione il vettore `[2, 4, 5, 4, 5, 5, 7, 9]`, questa ritornerebbe un puntatore ad un'area di memoria in grado di contenere i cinque interi `[2, 4, 5, 7, 9]` e imposterebbe la variabile puntata da `newsize` a `5`.
I puntatori passati non saranno mai NULL.

Esercizio 5 (8 punti)

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

```
extern void stampa_binario(const char* filename_in, const char* filename_out);
```

La funzione accetta in input due stringhe C. La prima contiene il nome di un file da aprire in modalità lettura non tradotta (binaria). Se il file esiste si deve creare in modalità scrittura tradotta (testo) un file utilizzando il nome passato come secondo parametro.
La funzione deve scrivere in output per ogni byte del file di input la sua rappresentazione in base 2 in formato testo utilizzando i caratteri '0' e '1' e dopo ogni byte scritto inserire uno spazio. Ogni 8 byte così scritti si deve inserire un carattere <a capo>.

Ad esempio, se abbiamo un file che contiene 2 byte che valgono 171 e 205 deve produrre un file contenente:

```
10101011 11001101
```

Se invece il file contenesse 10 byte che valgono 0,1,2,3,4,5,6,7,8,9, dovrebbe produrre:

```
00000000 00000001 00000010 00000011 00000100 00000101 00000110 00000111
00001000 00001001
```

Se il file di input non contiene byte, anche il file di output sarà vuoto. Se invece non è possibile aprire il file di input (ad esempio, perché non esiste) non si creerà neppure il file di output.