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)

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

```
extern double entropia(const double *v, size_t n);
```

La funzione riceve in input un puntatore a un vettore che contiene `n` elementi di tipo `double`, dove l'elemento di indice `i` rappresenta la probabilità $P(i)$ che una sorgente X emetta il valore i.

La funzione deve restituire l'entropia E della sorgente, ottenuta come:

$$E = -\sum_{i=0}^{n-1} P(i) \log_2 P(i)$$

Eventuali probabilità nulle non devono essere considerate nel calcolo dell'entropia.

Se `v` è `NULL`, `n=0`, o tutti gli elementi di `v` sono uguali a zero, la funzione restituisce `0`.

Esercizio 2 (6 punti)

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

```
extern bool easy_rle_decode(const char* nomefilein, const char* nomefileout);
```

La funzione accetta come parametro un nome di file di input (stringa C zero terminata) da aprire in modalità lettura non tradotta (binario) e un nome di file di output (stringa C zero terminata) da aprire in modalità scrittura non tradotta (binario).

Il file di input è codificato con coppie di byte `(n,c)`, il primo dei quali è con un numero intero senza segno. In output ogni coppia deve produrre `n+1` volte il byte `c`.

Ad esempio, dato il file seguente (visto come in un editor esadecimale):

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  00 61 01 62 02 63                                  .a.b.c
```

il file di output dovrebbe contenere

```
abbccc
```

La funzione ritorna `false` se non riesce ad aprire il file di input o di output.

Esercizio 3 (7 punti)

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

```
extern void cornice(uint8_t h, uint8_t w);
```

La funzione stampa su `stdout` una cornice rettangolare, costruita nel seguente modo.

Il lato superiore è ottenuto da `w` ripetizioni, separate dal carattere `|`, del pattern

```
/-\
```

Il lato inferiore è ottenuto da `w` ripetizioni, separate dal carattere `|`, del pattern

```
\-/
```

Il lato sinistro è ottenuto da `h` ripetizioni, separate dal carattere `-`, del pattern

```
/
|
\
```

Il lato destro è ottenuto da `h` ripetizioni, separate dal carattere `-`, del pattern

```
\
|
/
```

Ad esempio, invocando la funzione con `h=3` e `w=5`, essa deve stampare

```
/-\|/-\|/-\|/-\|/-\
|                 |
\                 /
-                 -
/                 \
|                 |
\                 /
-                 -
/                 \
|                 |
\-/|\-/|\-/|\-/|\-/
```

Se `h=0` o `w=0`, la funzione non stampa nulla.

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 *mirror_mat(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 parametri un puntatore ad una matrice `m` e deve restituire un puntatore a una nuova matrice allocata dinamicamente. La matrice è ottenuta specchiando la matrice di input in modo tale che la prima colonna diventi l'ultima, la seconda diventi la penultima, e così via. Se `m` è `NULL` la funzione restituisce `NULL`.

Ad esempio, data in input la matrice

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

la funzione restituisce la nuova matrice

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

Esercizio 5 (8 punti)

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

```
extern char *sottrai_stringhe(const char *a, const char *b);
```

La funzione riceve in input due puntatori a due stringhe C `a` e `b`, ciascuna delle quali contiene la rappresentazione ASCII in base 10 di un numero naturale composto da una quantità variabile di cifre. Il valore rappresentato da `a` sarà sempre maggiore o uguale a quello rappresentato da `b`.

La funzione deve restituire una stringa C che rappresenta la differenza tra `a` e `b`. Il risultato non deve contenere zeri superflui prima della cifra più significativa.

Ad esempio, con `a = "12345"` e `b = "12300"` la funzione deve restituire la stringa `"45"`. Con `a = "12345"` e `b = "4999"`, la funzione deve restituire la stringa `"7346"`.

Se `a` e `b` rappresentano lo stesso numero, la funzione restituisce `"0"`.

Se `a` o `b` valgono `NULL`, la funzione restituisce `NULL`.

Le stringhe di input saranno sempre correttamente formattate.