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 (6 punti)

Nel file `stringhe.c` implementare la definizione della seguente funzione:

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

La funzione accetta una stringa C (zero terminata) e ritorna una stringa C allocata dinamicamente sull'heap, contenente tutti i caratteri della stringa `s` presi una sola volta, nell'ordine con cui appaiono in `s`.
Ad esempio data la stringa `s="ciao casa"`, la funzione deve ritornare "`ciao s`".
Infatti, scorrendo `s`, incontriamo prima `'c'`, poi `'i'`, poi `'a'`, poi `'o'`, poi `' '`. Nessuno di questi è già stato visto. Poi incontriamo `'c'`, ma questo carattere è già stato incontrato e quindi non viene concatenato alla stringa di ritorno. Lo stesso succede per `'a'`, poi si incontra `'s'` e viene mandato in output. Infine troviamo ancora `'a'` e di nuovo lo ignoriamo.

Il puntatore ritornato deve puntare ad un'area di memoria grande esattamente il numero di byte necessari a contenere i caratteri unici e il terminatore. `s` non sarà mai NULL.

Esercizio 2 (6 punti)

Nel file `capovolgi.c` implementare la definizione della seguente funzione:

```
extern int capovolgi(const char *filein, const char *fileout);
```

La funzione accetta due nomi di file come stringhe C e deve aprire `filein` in lettura in modalità non tradotta (binaria) e `fileout` in scrittura in modalità non tradotta (binaria). La funzione deve copiare tutti i caratteri del file `filein` nel file `fileout`, ma in ordine inverso.
Ad esempio se il file `filein` contiene i seguenti byte (rappresentati in esadecimale nel seguito):
`AA BB FF AA BB CC`
il file `fileout` dovrà contenere:
`CC BB AA FF BB AA`
La funzione ritorna 0 se non riesce ad aprire uno dei due file, 1 altrimenti.

Esercizio 3 (punti 6)

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

```
struct matrix {
    size_t M, N;
    double *data;
};
```

e la funzione:

```
extern struct matrix *mat_replica(const struct matrix *a, int dir);
```

La struct consente di rappresentare matrici di dimensioni arbitraria, dove **`M` è il numero di righe, `N` è il numero di colonne** e `data` è un puntatore a `M×N` 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.M = 2`, `A.N = 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 parametro un puntatore a una matrice e deve ritornare una matrice allocata dinamicamente sull'heap ottenuta replicando la matrice in input orizzontalmente se dir=0, verticalmente altrimenti. Il puntatore alla matrice non sarà mai `NULL`.

Ad esempio, utilizzando la matrice A precedente, `mat_replica(A,0)` ritorna la matrice:

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

`mat_replica(A,1)` ritorna la matrice:

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

Esercizio 4 (7 punti)

Nel file istogramma.c implementare la definizione della seguente funzione:

```
extern void disegna(uint8_t* h, size_t n, FILE* fout);
```

La funzione accetta un puntatore ad un vettore di interi senza segno a 8 bit `h`, il numero di elementi presenti nel vettore `n` e un puntatore a file aperto in modalità scrittura tradotta (testo) `fout`.

Di seguito viene dettagliato il formato per disegnare un istogramma di interi utilizzando caratteri ASCII.
Dato un vettore di N interi, occorre disegnare N barre verticali in cui l'elemento in posizione `i` rappresenta l'altezza della `i`-esima barra verticale. Le barre verticali sono composte dal carattere `'|'`, gli spazi vuoti saranno riempiti col carattere `<spazio>`.

Ad esempio, dato il vettore di interi:
`1,2,3,4,5`
La rappresentazione come istogramma scritta su file sarà:

```
    |
   ||
  |||
 ||||
|||||
```

Il numero di righe scritte sul file dipenderà dal valore massimo presente nel vettore. Infatti, dato il vettore:
`1,2,3,1,2,3`
La rappresentazione come istogramma scritta su file sarà:

```
  |  |
 || ||
||||||
```

Composta quindi da 3 righe (perché il valore massimo è 3) di 6 caratteri ciascuna più un a capo (perché 6 è il numero di elementi).
Se un elemento vale 0 al posto della barra occorrerà scrivere uno `<spazio>`.
Ad esempio, dato il vettore:
`5,0,5,0,1,10`
La rappresentazione come istogramma scritta su file sarà:

```
    |
    |
    |
    |
    |
| |  |
| |  |
| |  |
| |  |
| | ||
```

Quindi con due spazi nel secondo e quarto elemento.
Il puntatore v non sarà mai NULL e punterà sempre ad almeno un elemento.

Esercizio 5 (8 punti)

Creare i file `animazione.h` e `animazione.c` che consentano di utilizzare le seguenti strutture:

```
#include <stdint.h> // Necessario per i tipi uint8_t e uint16_t

struct elem {
    uint16_t len;
    uint8_t *values;
};

struct animation {
    uint16_t num;
    struct elem *elems;
};
```

e la funzione

```
extern int anim_load(const char *filename, struct animation *anim);
```

È stato definito un formato binario di dati per memorizzare animazioni grafiche costituite da sequenze di elementi codificate in diversi modi. Un elemento è costituito da un campo `len` (intero senza segno a 16 bit codificato in little endian), seguito da `len` byte. Ogni animazione è costituita da uno o più elementi memorizzati uno dopo l'altro. Ad esempio il file `anim1.bin` contiene i seguenti byte (rappresentati in esadecimale nel seguito):
`03 00 01 00 02 02 00 03 04 05 00 FF CC AA EE DD`
Questa animazione contiene quindi un primo elemento di lunghezza 3, infatti i primi 2 byte sono `03 00`. I dati contenuti nell'elemento sono `01 00 02`.
Poi c'è un secondo elemento di lunghezza 2, infatti i successivi 2 byte sono `02 00`. I dati contenuti nell'elemento sono `03 04`.
Infine, c'è un terzo elemento di lunghezza 5, infatti i successivi 2 byte sono `05 00`. I dati contenuti nell'elemento sono `FF CC AA EE DD`.

La funzione `anim_load`, deve aprire il file il cui nome viene fornito dalla stringa C `filename` e caricarne il contenuto in memoria. La funzione deve

- Impostare il campo `num` della `struct animation` puntata da `anim` al numero di elementi presenti sul file
- Far puntare `elems` ad un vettore di `struct elem`, grande `num`, allocato dinamicamente sull'heap.
- Ogni elemento del vettore avrà il campo `len` impostato alla lunghezza dell'elemento e il campo `values` dovrà puntare ad un vettore di byte, grande `len`, allocato dinamicamente sull'heap, contenente i valori letti da file.

La funzione deve ritornare 1 se è riuscita ad aprire il file e a leggerne interamente il contenuto, 0 altrimenti. Tutti i file forniti (anim1.bin, anim2.bin, anim3.bin) esistono, contengono almeno un elemento e non hanno errori.