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 `rimuovi.c` implementare la definizione della seguente funzione:

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

La funzione accetta due nomi di file come stringhe C e deve aprire `filein` in lettura in modalità tradotta (testo) e `fileout` in scrittura in modalità tradotta (testo). La funzione deve copiare tutti i caratteri del file `filein` nel file `fileout`, riportando una volta sola i caratteri consecutivi ripetuti.
Ad esempio se il file `filein` contiene:
`abcdaae`
il file `fileout` dovrà contenere:
`abcdae`
ovvero le due lettere a consecutive sono state sostituite con una sola occorrenza.
La funzione ritorna 0 se non riesce ad aprire uno dei due file, 1 altrimenti.

Esercizio 2 (6 punti)

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

```
extern void itob(unsigned int x, char *sz, size_t n);
```

La funzione accetta un numero intero `x` e deve riempire la stringa C (deve essere zero terminata) all'indirizzo `sz` con la rappresentazione binaria di `x` a `n` bit in formato testo (ovvero i caratteri `'0'` e `'1'`). Ad esempio chiamando la funzione con `x=10` e `n=8` deve riempire la stringa `sz` con "`00001010`".
Il puntatore `sz` punta ad un'area di memoria già allocata e grande a sufficienza per contenere `n` caratteri più il terminatore.

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_sommadiretta(const struct matrix *a, const struct matrix *b);
```

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 }`.
L'operazione di somma diretta tra la matrice A di dimensioni $m \times n$ e la matrice B di dimensioni $p \times q$ è la matrice di dimensioni $(m+p) \times (n+q)$ definita come:

$$A \oplus B = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} = \begin{pmatrix} a_1^1 & \cdots & a_n^1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_1^m & \cdots & a_n^m & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_1^1 & \cdots & b_q^1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & b_1^p & \cdots & b_q^p \end{pmatrix}$$

Ad esempio:

$$\begin{pmatrix} 1 & 3 & 2 \\ 2 & 3 & 1 \end{pmatrix} \oplus \begin{pmatrix} 1 & 6 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 2 & 0 & 0 \\ 2 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

La funzione accetta come parametro due puntatori a matrici e deve ritornarne la somma diretta, allocata dinamicamente sull'heap. I puntatori alla matrice non saranno mai `NULL`.

Esercizio 4 (punti 7)

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

```
struct fraz {
    int num;
    unsigned int den;
};
```

e la funzione:

```
extern void fr_somma(struct fraz *ris, const struct fraz *a, const struct fraz *b);
```

Queste definizioni fanno parte di una libreria per il calcolo con i numeri razionali. I campi `num` e `den` di `struct fraz`, rappresentano rispettivamente il numeratore e il denominatore di una frazione.
La somma di frazioni, come noto, si può calcolare così:

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2}$$

La funzione `fr_somma` effettua la somma tra le due frazioni puntate da `a` e `b` e mette il risultato nella frazione puntata da `ris`. Il risultato deve essere ridotto ai minimi termini, ovvero numeratore e denominatore non devono avere divisori comuni oltre all'unità.
Per ottenere una frazione ai minimi termini, si possono dividere numeratore e denominatore per il loro massimo comune divisore.
Se ad esempio vogliamo calcolare $\frac{1}{2} + \frac{1}{2}$, con la formula precedente otteniamo $\frac{1 \cdot 2 + 1 \cdot 2}{2 \cdot 2} = \frac{4}{4}$. La funzione deve ritornare la frazione ridotta ai minimi termini, ovvero $\frac{1}{1}$.

Esercizio 5 (8 punti)

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

```c
#include <stdint.h> // Necessario per i tipi uint8_t e uint32_t

struct record {
    uint32_t size;
    uint8_t *data;
};

struct database {
    uint32_t num;
    struct record *recs;
};
```

e la funzione

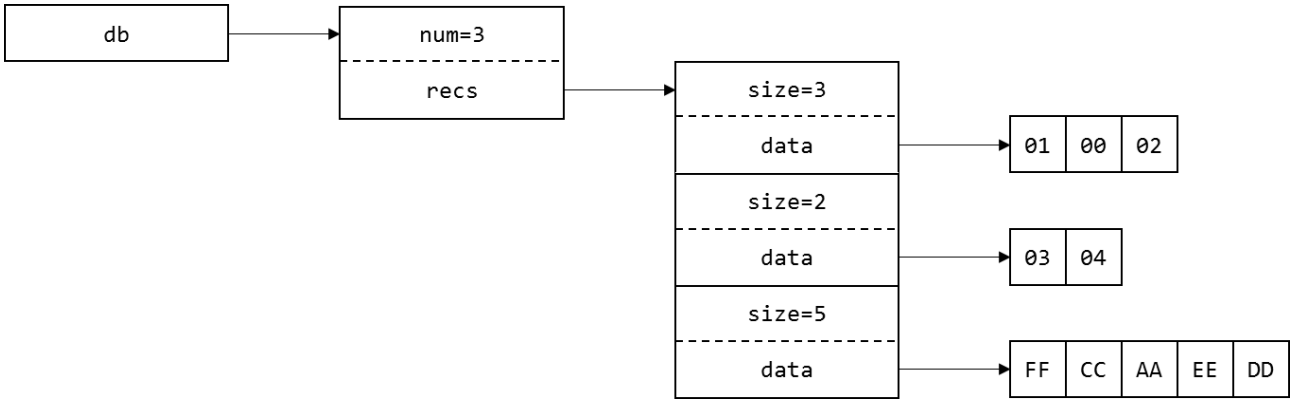
```c
extern int db_load(const char *filename, struct database *db);
```

È stato definito un formato binario di dati per memorizzare sequenze di informazioni codificate in qualsiasi modo, chiamate record. Un record è costituito da un campo `size` (intero senza segno a 32 bit codificato in little endian), seguito da `size` byte. Ogni database è costituito da uno o più record memorizzati uno dopo l'altro. Ad esempio il file `db1.bin` contiene i seguenti byte (rappresentati in esadecimale nel seguito):
`03 00 00 00 01 00 02 02 00 00 00 03 04 05 00 00 00 FF CC AA EE DD`
Il database contiene quindi un primo record di lunghezza 3, infatti i primi 4 byte sono `03 00 00 00`. I dati contenuti nel record sono `01 00 02`.
Poi c'è un secondo record di lunghezza 2, infatti i successivi 4 byte sono `02 00 00 00`. I dati contenuti nel record sono `03 04`.
Infine, c'è un terzo record di lunghezza 5, infatti i successivi 4 byte sono `05 00 00 00`. I dati contenuti nel record sono `FF CC AA EE DD`.

La funzione db_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 database puntata da db al numero di record presenti sul file
- Far puntare recs ad un vettore di struct record, grande num, allocato dinamicamente sull'heap.
- Ogni record del vettore avrà il campo size impostato alla lunghezza del record e il campo data dovrà puntare ad un vettore di byte, grande size, allocato dinamicamente sull'heap, contenente i dati letti da file.

Una visualizzazione grafica della memoria nel caso precedente sarebbe:



La funzione deve ritornare 1 se è riuscita ad aprire il file e a leggerne interamente il contenuto, 0 altrimenti. Tutti i file forniti (db1.bin, db2.bin, db3.bin) esistono, contengono almeno un record e non hanno errori.