Esame di Laboratorio del 17/02/2022

Esercizio 1 (5 Punti)

Scrivere nel file `sum.c` un programma a linea di comando (è necessario scrivere il `main` con i suoi parametri standard) con la seguente sintassi:

```
sum <value1> <value2> <value3> ...
```

Il programma prende in input da linea di comando un numero arbitrario di parametri e deve scrivere su `stdout` la somma dei parametri ricevuti seguita da un a capo, ignorando tutti i parametri che non sono numeri interi espressi come testo in base 10.

Ad esempio:

```
linea di comando      | stdout
----------------------+--------------
sum 1 2 3 -8          | -2↵
sum 1 a 2 b 3         | 6↵
sum 3.4 -22.87 c 3    | 3↵
sum                   | 0↵
sum ciao mondo!       | 0↵
```

Il programma termina sempre con codice di uscita `0` dopo aver completato la scrittura.

Esercizio 2 (6 punti)

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

```c
struct matrix {
    size_t rows, cols;
    double *data;
};
```

e la funzione:

```c
extern struct matrix *mat_rowmul(const struct matrix *m1, const struct matrix *m2);
```

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 parametro due puntatori alle matrici `m1` e `m2`, e deve restituire un puntatore a una nuova matrice allocata dinamicamente. `m2` deve avere tante righe quante `m1` e una sola colonna (è in pratica un vettore colonna). La nuova matrice è ottenuta moltiplicando ogni elemento di una riga di `m1` per l'elemento alla riga corrispondente di `m2`.

Se `m1` è `NULL`, oppure se `m2` è `NULL`, oppure se la moltiplicazione per righe non è applicabile tra le due matrici, la funzione restituisce `NULL`.

Ad esempio, indicando l'operazione sopra descritta con $\circledS$, si ha che :

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \circledS \begin{pmatrix} 7 \\ 8 \end{pmatrix} = \begin{pmatrix} 7 & 14 & 21 \\ 32 & 40 & 48 \end{pmatrix}$$

Esercizio 3 (7 punti)

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

```
struct angolo {
    uint16_t gradi;
    uint8_t primi;
    uint8_t secondi;
};
```

e la funzione:

```
extern struct angolo differenza_angoli(struct angolo a, struct angolo b);
```

La `struct angolo` consente di rappresentare un angolo in termini di gradi, primi e secondi.

- Un `grado` è la 360-esima parte di un angolo giro (simbolo °).
- Un `primo` è la 60-esima parte di un grado (simbolo ').
- Un `secondo` è la 60-esima parte di un primo (simbolo '').

In altre parole, 60 secondi formano un primo, 60 primi formano un grado e 360 gradi formano un angolo giro.

Ad esempio, la struct:

```
{ 230, 47, 12 }
```

Rappresenta l'angolo 230° 47' 12''.

La misura di un angolo è detta in **forma normale** se gradi, primi e secondi sono non negativi e i gradi sono minori o uguali a 360, mentre i primi e i secondi sono minori di 60.

La funzione `differenza_angoli` accetta come parametri due angoli e ne restituisce la differenza, espressa in forma normale. Se il primo angolo è minore del secondo, la differenza non si può fare e la funzione ritorna un valore convenzionale per indicare che l'angolo non è valido, ovvero `{ 0xFFFF, 0xFF, 0xFF }`.

Ad esempio, invocando la funzione con:

```
```

```
a = { 230, 23, 4 }
b = { 45, 47, 12 }
```

Essa deve restituire:

```
{ 184, 35, 52 }
```

Invertendo invece gli angoli, il risultato sarebbe:

```
{ 0xFFFF, 0xFF, 0xFF }
```

Gli angoli passati alla funzione saranno sempre validi e già in forma normale.

Esercizio 4 (7 Punti)

Creare i file `dataset.h` e `dataset.c`, che consentano di usare le struct:

```c
struct row {
    char *filename; // max 255 characters
    char prognosis; // M or S
};

struct dataset {
    struct row *data;
    size_t nrows;
};
```

e la funzione:

```c
extern struct dataset *dataset_load(const char *filename);
```

Un dataset è un insieme di casi clinici composti da una radiografia toracica e da una prognosi della malattia. Le informazioni sul dataset sono salvate in un file di testo composto da un nome di file (lunghezza massima 255 caratteri, non può contenere virgole), una virgola, una prognosi (che può essere "MILD" o "SEVERE"), un carattere `\n`.

```
P_3_29.png,MILD↵
P_3_415.png,MILD↵
P_3_163.png,MILD↵
P_3_348.png,SEVERE↵
P_3_263.png,SEVERE↵
```

La funzione deve aprire il file in modalità non tradotta (binaria), leggere i casi e per ogni riga inserire una `struct row` in un dataset. Tutto deve essere allocato dinamicamente. Nelle righe bisogna inserire un puntatore al nome del file (allocato dinamicamente) e il carattere `M` o `S` per la prognosi.

Se non è possibile aprire il file la funzione ritorna NULL. Il file rispetterà sempre il formato indicato (non servono controlli).

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
        ^--------^---- spazi (questa riga non è nel file!)
```

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.