Esame di Laboratorio del 02/02/2022

Esercizio 1 (5 Punti)

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

```
lower <filename>
```

Il programma prende in input da linea di comando un parametro di tipo stringa `filename`, e deve scrivere su `stdout` tutto il contenuto del file, ma le lettere maiuscole (del set di caratteri standard ASCII) devono diventare minuscole.

Se il numero dei parametri passati al programma non è corretto o se non è possibile aprire il file, il programma termina con codice `1`, senza scrivere nulla su `stdout`, altrimenti termina con codice di uscita `0` dopo aver completato la scrittura.

Esercizio 2 (6 punti)

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

```
extern void scrivi_v(FILE *f, uint8_t n);
```

La funzione scrive sul file `f`, già aperto in scrittura in modalità tradotta (testo), una lettera `V` maiuscola, costruita nel seguente modo:

- il lato obliquo sinistro è composto da `n` caratteri `\`.
- il lato obliquo destro è composto da `n` caratteri `/`.
- dopo ogni `/` la funzione deve scrivere un a capo (indicato graficamente con ↵ nel seguito).

Con `n=0` non scrive nulla. Con `n=1` scrive "`\/↵`", con `n=2`:

```
\  /↵
 \/↵
```

e così via. Ad esempio, invocando la funzione con `n=5`, la funzione deve stampare

```
\        /↵
 \      /↵
  \    /↵
   \  /↵
    \/↵
```

Spazi e a capo devono essere esattamente quelli indicati negli esempi.

Esercizio 3 (7 punti)

Creare il file `memmem.c` che consenta di utilizzare la seguente funzione:

```
extern const void *memmem(const void *pagliaio, size_t psize, const void *ago, size_t asize);
```

La funzione `memmem` accetta come parametri due puntatori a zone di memoria `pagliaio` e `ago`, e le dimensioni di esse espresse in byte `psize` e `asize`. La funzione cerca `ago` in `pagliaio` e restituisce un puntatore alla prima posizione di un

blocco di memoria uguale ad `ago`, all'interno del blocco di memoria `pagliaio`. Se l'ago non è presente nel `pagliaio`, se `pagliaio` è `NULL`, se `psize` è `0`, se `ago` è `NULL`, o se `asize` è `0`, la funzione restituisce `NULL`.

Consiglio

Al fine di svolgere questo esercizio, valutate di usare le funzioni della libreria standard `memchr()` e `memcmp()`.

Esercizio 4 (7 punti)

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

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

e la funzione:

```
extern struct matrix *mat_mul(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. La nuova matrice è ottenuta effettuando il prodotto righe per colonne tra le matrici `m1` e `m2`.

Siano date una matrice A di dimensione $m \times n$ ed una seconda matrice B di dimensioni $n \times p$. Siano a_{ij} gli elementi di A e b_{ij} gli elementi di B. Si definisce il prodotto matriciale di A per B la matrice $C = AB$ di dimensioni $m \times p$ i cui elementi c_{ij} sono dati da:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

per ogni valore di riga i e di colonna j.

Due matrici possono essere moltiplicate fra loro solo se il numero di colonne della prima è uguale al numero di righe della seconda, e il prodotto tra due matrici non è commutativo.

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

Ad esempio, data la matrice:

$$m1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

e la matrice

$$m2 = \begin{pmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{pmatrix}$$

La funzione restituisce la nuova matrice:

$$\begin{pmatrix} 50 & 68 \\ 122 & 167 \end{pmatrix}$$

ottenuta con il prodotto righe per colonne:

$$\begin{pmatrix} 1 \cdot 7 + 2 \cdot 8 + 3 \cdot 9 & 1 \cdot 10 + 2 \cdot 11 + 3 \cdot 12 \\ 4 \cdot 7 + 5 \cdot 8 + 6 \cdot 9 & 4 \cdot 10 + 5 \cdot 11 + 6 \cdot 12 \end{pmatrix}$$

Esercizio 5 (8 Punti)

Una delle sfide del gioco di società Cortex[2] è Unico: data una carta contenente disegni di oggetti, bisogna trovare quello che è unico nella forma o nel colore. Nell'esempio qui sopra è unica la nota di colore giallo.

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

```
struct oggetto {
    char *forma;
    char *colore;
};

struct carta {
    struct oggetto *disegni;
    size_t n;
};
```

e la funzione:

```
extern const struct oggetto *unico(const struct carta *c);
```

La funzione deve restituire l'indirizzo dell'oggetto contenuto in `c` che è unico per forma o per colore. Può esistere un altro oggetto con lo stesso colore dell'oggetto che è unico per forma e può esistere un altro oggetto con la stessa forma dell'oggetto che è unico per colore.

La forma sarà una stringa C tipo "matita", "missile", "nota", "freccia", ... e il colore sarà una stringa C tipo "rosso", "blu", "verde", ... Ogni carta contiene sempre un oggetto unico o per forma o per colore. Ogni carta contiene sempre più di un oggetto. Tutti i puntatori sono sempre validi e non servono controlli.

Ecco un main per testare la funzione:

```
int main(void)
{
    struct carta c = {
        (struct oggetto[]) {
            { "missile", "verde" },
            { "bomba", "verde" },
            { "bomba", "rosso" },
            { "missile", "nero" },
            { "lampadina", "blu" },
            { "nota", "giallo" },
            { "cavallo", "rosso" },
            { "lampadina", "nero" },
            { "bomba", "blu" },
            { "cavallo", "giallo" },
        }, 10 };
    const struct oggetto *o = unico(&c);
    return 0;
}
```

o dovrebbe puntare a `{ "nota", "giallo" }`, ovvero `c->disegni[5]`.