Esame di Laboratorio del 23/01/2023

Esercizio 1 (5 Punti)

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

```
fizzbuzz <numero_intero_positivo>
```

Il programma prende in input da linea di comando un parametro di tipo int (chiamiamolo `n`) e scrive su `stdout` tante righe (da `1` a `n`):

1. se l'indice di riga è divisibile per 3 ma non per 5 scrive "Fizz"
2. se l'indice di riga è divisibile per 5 ma non per 3 scrive "Buzz"
3. se l'indice di riga è divisibile per 3 e per 5 scrive "Fizzbuzz"
4. altrimenti scrive il numero stesso in decimale.

Se il numero di parametri non è corretto o il parametro è errato, il programma non scrive niente e termina con codice di uscita `1`, altrimenti termina con codice di uscita `0`.

Ad esempio chiamando il programma con `n = 20` si ottiene:

```
1↵
2↵
Fizz↵
4↵
Buzz↵
Fizz↵
7↵
8↵
Fizz↵
Buzz↵
11↵
Fizz↵
13↵
14↵
Fizzbuzz↵
16↵
17↵
Fizz↵
19↵
Buzz↵
```

Esercizio 2 (6 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_normalize_rows(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 parametro un puntatore alla matrice `m` e deve restituire un puntatore a una nuova matrice allocata dinamicamente costruita in modo che le righe della matrice siano vettori a norma 1 con la stessa direzione e verso di quelli della matrice di input. Per fare questo, si deve dividere ogni elemento di ogni riga della matrice di input per la norma euclidea della riga stessa. Indicando con $r_i = (a_{i1}, \ldots, a_{in})$ il vettore composto dagli elementi della riga i-esima, bisogna produrre in output una riga $\hat{r}_i$ con:

$$\hat{r}_i = \frac{r_i}{||r_i||}$$

Se il vettore r_i ha norma 0, la riga non deve essere modificata (non si potrebbe infatti dividere per 0).

La norma euclidea di un vettore n-dimensionale x è definita come:

$$||x|| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Alcuni esempi (i valori sono arrotondati alla quarta cifra decimale):

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 0.2673 & 0.5345 & 0.8018 \\ 0.4558 & 0.5698 & 0.6838 \end{pmatrix}$$

$$\begin{pmatrix} 7 & 5 \\ 3 & 12 \\ 0 & 0 \\ 11 & 8 \end{pmatrix} \rightarrow \begin{pmatrix} 0.8137 & 0.5812 \\ 0.2425 & 0.9701 \\ 0 & 0 \\ 0.8087 & 0.5882 \end{pmatrix}$$

Esercizio 3 (7 Punti)

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

```
extern void translate(char *str, const char *from, const char *to);
```

La funzione deve cercare ogni carattere di `str` in `from` e se il carattere è presente, sostituirlo con l'elemento di posizione corrispondente in `to`. Ad esempio, data la stringa `s = "ciao"`, chiamare la funzione `translate(s, "abdc", "wxzy")` trasforma `s` in `"yiwo"`.

Se qualsiasi parametro di input è `NULL` o se `from` e `to` hanno lunghezze diverse, la funzione non modifica `str`.

Esercizio 4 (7 Punti)

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

```
extern size_t shorten(int *v, size_t n, int max);
```

La funzione riceve un vettore `v` di `n` int (interi a 32 bit con segno) e deve rimuovere tutti gli elementi maggiori di `max`, senza modificare l'ordine degli altri. La funzione ritorna la nuova dimensione ottenuta dopo la rimozione degli elementi. Se `v` è `NULL`, la funzione ritorna 0.

Ad esempio dato il vettore `x = { 5, 1, 7, 9, 11, 3, 8, 2, 1, 3, 5 }` di 11 elementi, eseguendo la funzione con:

```
size_t new_size = shorten(x, 11, 7);
```

`x` diventa `{ 5, 1, 7, 3, 2, 1, 3, 5 }` e `new_size` assume il valore 8.

Ovviamente la funzione non può modificare l'indirizzo di memoria di `x`, quindi non fatevi venire in mente di usare `realloc` per "accorciare" lo spazio occupato. Questo sarà, eventualmente, compito di chi utilizza la funzione.

Esercizio 5 (8 punti)

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

```
struct city {
    char *name; // max length = 255
```

```
    uint32_t population;
};
```

e la funzione:

```
extern struct city *read_cities(const char *filename, uint32_t *n);
```

La struct consente di rappresentare città con un nome di dimensione arbitraria (ma minore di 255), e un numero intero a 32 bit senza segno che ne indica la popolazione.

È definito un formato di file binario che consente di rappresentare una sequenza di città ognuna con la corrispondente popolazione. Il file comincia con un numero intero a 32 bit senza segno che indica il numero n di elementi nel file. Di seguito ci sono n coppie di città e popolazione. Ogni città è rappresentata come una sequenza di caratteri ASCII zero terminata, seguita da un numero intero a 32 bit senza segno in little endian (la popolazione). Ad esempio il file binario seguente (visualizzato come in un editor esadecimale):

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  03 00 00 00 4D 6F 64 65 6E 61 00 D9 CF 02 00 42  ....Modena.ÙÏ..B
00000010  6F 6C 6F 67 6E 61 00 0F EB 05 00 50 65 63 68 69  ologna..ë..Pechi
00000020  6E 6F 0A 15 76 01                                 no..v.
```

contiene 3 città: Modena (184 281 abitanti) Bologna (387 855 abitanti) Pechino (24 515 850 abitanti)

La funzione deve aprire il file `filename` in modalità lettura non tradotta (binario) e allocare una sequenza di `struct city` di tanti elementi quanto indicato dal primo valore letto dal file. Ogni elemento conterrà il nome della città come stringa C (allocato dinamicamente e zero terminato) e la popolazione. La funzione ritorna un puntatore alla sequenza prodotta e imposta la variabile puntata da n al numero di città presenti nel file.

Se l'apertura del file fallisce, o se non è possibile leggere correttamente il contenuto del file, la funzione non alloca memoria e ritorna `NULL`, senza modificare la variabile puntata da n.