Esame di Laboratorio del 09/01/2023

Esercizio 1 (5 Punti)

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

```
ternary2decimal <sequenza senza spazi di -, 0, +>
```

Il programma prende in input da linea di comando un parametro di tipo stringa composto da una sequenza di `-`, `0` e `+` che rappresentano rispettivamente i valori $-1, 0, +1$. La sequenza è un numero intero espresso in ternario bilanciato, ovvero un numero in base 3, in cui il valore della cifra meno significativa (quella più a destra) è moltiplicata per 3^0, la successiva (una posizione più a sinistra) per 3^1 e così via.

La stringa `+++` corrisponde al numero $(+1)\cdot 3^2 + (+1)\cdot 3^1 + (+1)\cdot 3^0 = 3^2 + 3^1 + 3^0 = 9 + 3 + 1 = 13$.

La stringa `+0-` corrisponde al numero $(+1)\cdot 3^2 + (0)\cdot 3^1 + (-1)\cdot 3^0 = 3^2 - 3^0 = 9 - 1 = 8$.

La stringa `-0+` corrisponde al numero $(-1)\cdot 3^2 + (0)\cdot 3^1 + (+1)\cdot 3^0 = -3^2 + 3^0 = -9 + 1 = -8$.

Il programma deve scrivere su `stdout` il valore del numero ricevuto in input in base 10 **e un a capo**. Se il numero di parametri non è corretto o il parametro contiene caratteri diversi da meno, zero o più, il programma non scrive niente e termina con codice di uscita `1`, altrimenti termina con codice di uscita `0`.

Il valore sarà sempre rappresentabile in un `int` (non servono controlli).

Ad esempio:

```
linea di comando        | stdout
------------------------+--------------
+++                     | 13↵
+0-                     | 8↵
-0+                     | -8↵
+00-+0++-+-00-          | 1557683↵
+010+                   |               (caratteri non validi)
++ --                   |               (troppi parametri)
                        |               (nessun parametro)
```

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 void mat_normalize_rows(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 modificarne i valori in modo che le righe della matrice siano vettori a norma 1. Per fare questo, si deve dividere ogni elemento di ogni riga della matrice 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 sostituire quella riga con:

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

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

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

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

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

Esercizio 3 (7 Punti)

È disponibile un file con un elenco dei valori di alcuni oggetti, uno per riga. Questi sono divisi in gruppi separati da una sola riga vuota. Ad esempio:

```
1000
2000
3000

4000

-5000
6000

7000
-6000
-9000

10000
```

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

```c
extern int *read_gruppi(const char *filename, size_t *ngruppi);
```

La funzione deve aprire in modalità lettura tradotta (testo) il file `filename` e restituire una sequenza di `int` allocata dinamicamente in memoria, contenente in ogni elemento la somma dei valori di ogni gruppo. Inoltre la funzione imposta la variabile puntata da `ngruppi` al numero di gruppi letti.

Nell'esempio precedente, la funzione restituirebbe una sequenza contenente i valori `{ 6000, 4000, 1000, -8000, 10000 }` e la variabile puntata da `ngruppi` verrebbe impostata a `5`.

Se la funzione non riesce ad aprire il file ritorna `NULL` e non modifica la variabile puntata da `ngruppi`.

I file passati alla funzione saranno sempre corretti (non servono controlli), al termine del file può esserci o meno un a capo dopo l'ultimo valore, e i valori e le loro somme saranno sempre rappresentabili con un `int`.

Esercizio 4 (7 Punti)

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

```
extern void plotter(const char *p);
```

La funzione riceve un vettore di char (interi a 8 bit con segno) e lo scorre. Ad ogni passo:

- se il valore corrente è negativo, invia su `stdout`, tante volte quanto il suo valore assoluto, l'elemento successivo e avanza di due posizioni;
- se il valore corrente è positivo, invia su `stdout` quel valore e avanza di una posizione;
- se il valore corrente è zero, termina.

Ad esempio, consideriamo questo array di 15 char:

```
char plus[] = { -10,' ','*','\n',-9,' ',-3,'*','\n',-10,' ','*','\n',0 };
```

La funzione `plotter()` scriverà 10 spazi e avanza di due, poi un asterisco e avanza di uno e poi un a capo e avanza di uno. Nella seconda riga ci saranno 9 spazi, 3 asterischi e l'a capo. Infine nuovamente 10 spazi, asterisco e a capo. Poi lo zero termina l'esecuzione.

Ecco come potrebbe essere utilizzata:

```
extern void plotter(const char *program);

int main(void)
{
    char plus[] = { -10,' ','*','\n',-9,' ',-3,'*','\n',-10,' ','*','\n',0 };
    plotter(plus);
    char square[] = { -5,'*','\n','*',-3,' ','*','\n','*',-3,' ','*','\n','*',-3,' ','*',
                      '\n',-5,'*','\n',0};
    plotter(square);
    plotter("\366 *\n\367 \375*\n\366 *\n"); // This is another version of plus!
    plotter("\373 \373_\362 .\375-\375.-.\n\375 ,'\376 -. `.\366 ,' \373_\375.'\n\376 /\375"
            " - _ - \\\370 : .' _\375 _ \376\\\n :\374 ' _)'\376 :\371 | :-(_).(_)\376:\n(_"
            "\365 ;)\372 | |\374 -'\376 \376|\n\376 \\\373 _\375 /\370 ; |\374 _\375 \376|\n"
            "\375 `\376.\375_\376.'\367 `-'\376.\374_.'`'\n\372 ;._:\361 _; :_\n\373 /\374 "
            "\\ \370 ,'\376 `' `. \n\n");
    plotter("\355 YAao,\n\354 Y\3748b,\n\356 ,oA\3718b,\n\364 ,\375ad\3608bo,\n\367 ,d\345"
            "8b,\n\371 ,\3378b,\n\372 d\3338,\n\373 d\3318b\n\374 d\3728P'\354 `Y\3648,\n"
            "\374 \3738P'\354 Yb\374a\36681\n\375 a\3748'\352 `Y\3748P' `V\3728\n d\3718a"
            "\340 `Y\3748\nAY/\376' `\\Y8b\337 \376`Y8b\nY'\372 `YP\334 \376~\n\367 `'\n\n");
    plotter("\362 .*\352\"*.\n\363 :\346 ;\n\363 :\346 ;\n\363 :\376 \352.\376 ;\n\363 : :"
            "\352 ; ;\n\363 :_:\352 ;_;\n\364 /\376 :\376 \376_\375.\370-\375.\376_\376 ;"
            "\376 \\\n\365 /\375 :\"'\376 .*\"*-.\376 .-*\"*.\376 '\";\375 \\\n\366 :\374 "
            ";\375 /\372 ;:\372 \\\375 :\374 ;\n\366 !\374 !\376 ;\374 *\376 \376!\376 *"
            "\374 :\376 !\374 !\n\366 ;\375 ;\375 :\373 .'\376 '.\373 ;\375 ;\375 :\n\366 "
            ":\376 .'\374 '-.-'\372 '-.-'\374 '.\376 ;\n\366 '-\"\\\346 /\"-'\n\362 '.\352"
            " .'\n\360 *,\372 '-\376_-'\372 ,*\n\360 /.'-_\364 _-' .\\\n\361 /\376 \"-_\"*"
            "-.\374_.-*\" _-\"\376 \\\n\362 /\372 '-_\376 /'\\\376 _-'\372 \\\n\363 :\374 "
            ":\375 \376_'\" | \"'\376_\375 ;\374 ;\n\363 |.\376-.;\376 |\\/|\376 |\376 |\\"
            "/|\376 :.\376-.|\n\363 (\375 ()\376 |\376_|\376 |\376 |\376_|\376 ()\375 )\n"
            "\362 '\376-^_\370 |\370 _^\376-'\n\357 | \"'*\376-\376._I_\376.\376-*'\" |\n"
            "\357 | \376_\375._\376 | _\375._\375 |\n\360 .'\"\372 `\"'\"\373 \376'\"'.\n"
            "\360 \n\n");
    return 0;
}
```

Esercizio 5 (8 punti)

Il sistema imperiale inglese esprime le lunghezze in termini di inches (pollici), feet (piedi), yards (iarde) e miles (miglia). Un piede è 12 pollici, una iarda è 3 piedi e un miglio è 1760 iarde. Un pollice è 0,0254 m.

Nella scrittura si può utilizzare:

- pollici: `inch` (singolare), `inches`, `in`, `in.`, `"` (le doppie virgolette, senza spazio prima)
- piedi: `foot` (singolare), `feet`, `ft`, `ft.`, `'` (singolo apice, senza spazio prima)
- iarde: `yard` (singolare), `yards`, `yd`, `yd.`
- miglia: `mile` (singolare), `miles`, `mi`, `mi.`

Esempi:

- Los Angeles is 2789.97 mi away from New York by car
- A football field is 100 yd. long.
- The wall was 9 ft. long.
- Ritchson matches Reacher's height: 6' 5".
- The spider was 1 in. across.

Creare i file `imperial.h` e `imperial.c` che consentano di utilizzare la seguente funzione:

```
extern double to_meter(const char *imperial_length);
```

La funzione `to_meter` accetta come parametro una stringa contenente una misura di lunghezza espressa come testo secondo il sistema imperiale inglese e la restituisce convertita in metri. La misura è composta da un numero (con o senza decimali) seguito da una delle unità di misura precedentemente elencate, eventualmente più volte (separate da spazi).

Ad esempio (i valori sono arrotondati alla quarta cifra decimale):

`imperial_length`	`to_meter`
`1 inch`	0.0254
`2789.97 mi`	4490021.4797
`6' 5"`	1.9558 (altezza di Jack Reacher)
`100 yd.`	91.4400 (lunghezza del campo da football americano)
`26 miles 385 yards`	42194.9880 (lunghezza di una maratona)

Le stringhe passate alla funzione saranno sempre corrette (non servono controlli).

SUGGERIMENTO: partite facendone una versione che supporta solo un numero e una unità di misura, estendendo solo dopo a più elementi consecutivi. La funzione `sscanf` può essere utile, se sapete usare lo specificatore `%n`.