Compito 09/09/2020

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 (5 punti)

Una sequenza di interi è detta *aritmetica* se contiene almeno due interi e tutte le differenze tra interi consecutivi sono uguali. Per esempio [9, 10], [3, 3, 3], e [9, 7, 5, 3] sono sequenze aritmetiche, mentre [1, 3, 3, 7], [2, 1, 2], e [1, 2, 4] non lo sono.

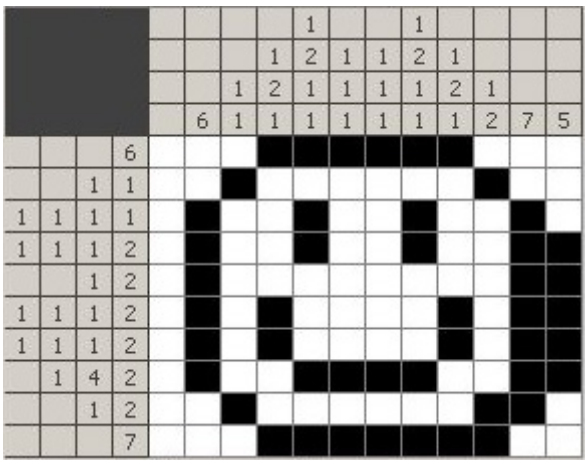
Nel file `arithmetic.c` implementare la definizione della funzione:

```
extern bool is_arithmetic(const int *v, size_t n);
```

La funzione riceve in input un vettore `v` contenente una sequenza di `n` interi, e deve ritornare `true` se la sequenza è aritmetica, `false` altrimenti. Se `v` è `NULL`, la funzione ritorna `false`.

Esercizio 2 (6 punti)

La parola *Nonogram* identifica una classe di rompicapi logici grafici, utili a svelare un'immagine nascosta, in cui le celle di una griglia devono essere colorate o lasciate in bianco in base a dei numeri a lato della griglia. In questo tipo di rompicapo, gli indizi vengono presentati per mezzo di numeri che indicano quante celle consecutive devono essere riempite, per riga o per colonna. Ad esempio un indizio di riga del tipo `1 4 2` indica che la riga corrispondente conterrà tre insiemi di rispettivamente uno, quattro e due caselle da riempire in questo ordine, con almeno una casella bianca tra gruppi successivi.



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

```
extern uint8_t* nonogram_row(const char* s, size_t *n);
```

La funzione riceve in input una stringa C `s` e un puntatore ad intero a 32 bit `n`. La stringa `s` contiene soltanto i caratteri `' '` (spazio) e `'*'` (asterisco), e rappresenta una riga di un nonogram. Il carattere `' '` corrisponde ad una cella vuota, mentre il carattere `'*'` indica una cella colorata. La funzione deve ritornare il puntatore ad un nuovo vettore, allocato dinamicamente, contenente il corrispondente indizio di riga, nella forma di una sequenza di interi a 8 bit senza segno. La dimensione del vettore creato deve essere scritta in memoria all'indirizzo a cui punta `n`.

Ad esempio, data in input la stringa `" * **** **"`, la funzione deve restituire il vettore `{1,4,2}`, e la zona di memoria a cui punta `n` deve contenere il valore `3`.

Se `s` vale `NULL`, la funzione ritorna `NULL`, e non modifica il valore a cui punta `n`.

Esercizio 3 (7 punti)

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

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

e la funzione:

```
extern double* matrix_snake(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 `matrix_snake` accetta come parametro un puntatore a matrice `m`, e restituisce un puntatore ad un nuovo vettore di `double`, allocato dinamicamente.

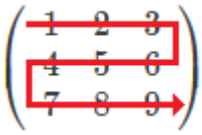
Il vettore da ritornare deve contenere tutti gli elementi della matrice, nell'ordine seguente:

- la prima riga, da sinistra a destra (→)
- la seconda riga, da destra a sinistra (←)
- la terza riga, da sinistra a destra (→)
- la quarta riga, da destra a sinistra (←)

- ecc... Ad esempio, data la matrice:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

L'ordine da considerare è:



La funzione, quindi, deve ritornare il vettore:

$$V = \begin{pmatrix} 1 & 2 & 3 & 6 & 5 & 4 & 7 & 8 & 9 \end{pmatrix}$$

Se `m` è `NULL`, la funzione restituisce `NULL`.

Esercizio 4 (7 punti)

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

```
extern char* change_font(const char *s);
```

La funzione `change_font` accetta come parametro un puntatore ad una stringa C contenente una qualsiasi stringa contenente lettere unicamente maiuscole, e ne ritorna la versione modificata, allocata dinamicamente su heap.

La funzione deve sostiture caratteri o brevi sequenze di caratteri come riportato di seguito, effettuando prima le sostituzioni che compaiono per prime nella tabella:

```
SEI ->  6
PER ->  X
A   ->  4
E   ->  3
I   ->  1
O   ->  0
S   ->  5
```

Tutti i caratteri non presenti nella tabella non vengono modificati

Ad esempio, data in input la stringa:

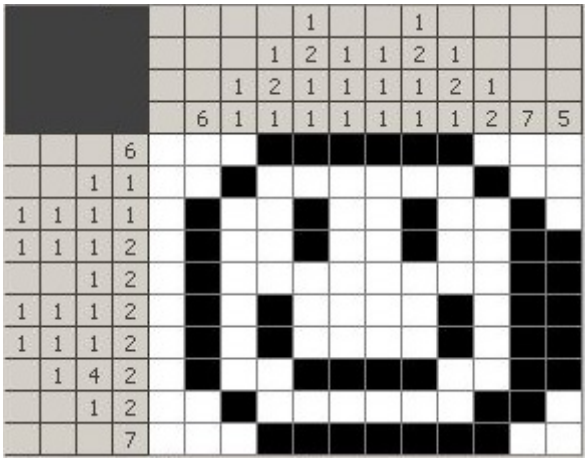
```
"QUATTRO PAPERE VISITARONO SEI MUSEI."
```

la funzione restituisce `"QU4TTR0 P4X3 V151T4R0N0 6 MU6."`.

La stringa `s` è sempre formattata correttamente, se `s` vale `NULL` la funzione restituisce `NULL`.

Esercizio 5 (8 punti)

La parola *Nonogram* identifica una classe di rompicapi logici grafici, utili a svelare un'immagine nascosta, in cui le celle di una griglia devono essere colorate o lasciate in bianco in base a dei numeri a lato della griglia. In questo tipo di rompicapo, gli indizi vengono presentati per mezzo di numeri che indicano quante celle consecutive devono essere riempite, per riga o per colonna. Ad esempio un indizio di riga del tipo `1 4 2` indica che la riga corrispondente conterrà tre insiemi di rispettivamente uno, quattro e due caselle da riempire in questo ordine, con almeno una casella bianca tra gruppi successivi.



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

```
struct nonogram {
    size_t n;
    char* schema;
};
```

e la funzione:

```
extern bool nonogram_load(struct nonogram* ng, const char* filename);
```

La `struct nonogram` consente di rappresentare l'immagine di un nonogram quadrato di lato `n`, in modo analogo ad una matrice. Il campo `schema` è il puntatore ai dati: è allocato dinamicamente, ha dimensione $n \times n$ e contiene l'immagine riga per riga. Le celle bianche sono rappresentate dal carattere `' '` (spazio), mentre le celle colorate sono simboleggiate dal carattere `'*'` (asterisco).

La funzione `nonogram_load()` accetta un puntatore ad una `struct nonogram` già allocata, e la popola con i dati caricati dal file di testo `filename`, che deve essere aperto in modalità tradotta, ed è strutturato come segue:

```
<n><a capo>
<prima riga><a capo>
<seconda riga><a capo>
ecc...
```

Ad esempio:

```
4
****
*  *
*  *
****
```

È compito della funzione allocare lo spazio necessario per il campo `schema`. Il file sarà sempre formattato correttamente. Se il file non esiste, o è impossibile aprirlo, la funzione ritorna `false`; diversamente, ritorna `true`.