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 (6 punti)

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

```
extern void print_board(FILE *f, const unsigned short board[4][4]);
```

La funzione prende come input un puntatore ad un FILE aperto in modalità tradotta (testo) e un puntatore a 4 array di 4 unsigned short. La funzione deve inviare sul file formattata la tavola del gioco, riportando in una griglia i valori numerici (che saranno sempre minori o uguali a 2048). Se il valore è 0, non bisogna scrivere nulla nella griglia, altrimenti bisogna scrivere il numero allineato a destra.

Ad esempio, dato l'array con i valori

```
256     512     512     2
128     32      32      1024
2       0       128     128
32      256     64      128
```

l'output dovrà essere:

```
+----+----+----+----+↲
| 256| 512| 512|   2|↲
+----+----+----+----+↲
| 128|  32|  32|1024|↲
+----+----+----+----+↲
|   2|    | 128| 128|↲
+----+----+----+----+↲
|  32| 256|  64| 128|↲
+----+----+----+----+↲
```

Esercizio 2 (5 punti)

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

```
extern bool is_tga(const char *filename);
```

La funzione accetta in input una stringa C che contiene il nome di un file e verifica se il file contiene un immagine nel formato TGA (versione 2), verificando che gli ultimi 18 byte del file contengano i caratteri `TRUEVISION-XFILE.` e un carattere NUL (un byte a 0). Se il file non esiste, se non è possibile leggere 18 byte o se i 18 byte letti non corrispondono alla sequenza fornita, la funzione ritorna `false`, altrimenti `true`.

Suggerimento: utilizzate HxD per aprire gli esempi forniti e verificare quali file rispettano effettivamente il formato TGA.

Esercizio 3 (7 punti)

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

```
extern void reverse(int *vec, size_t len, size_t from, size_t to);
```

La funzione accetta come parametri un puntatore ad un vettore `vec` di int e la relativa lunghezza `len`, e due indici `from` e `to`. La funzione deve ribaltare tutti i valori dalla posizione `from` fino alla posizione `to` **esclusa**.

Ad esempio dato il vettore
`43, 14, 72, 48, 36, 33, 46, 75, 50, 86`
chiamando la funzione con `from=2` e `to=7` il vettore dovrà contenere
`43, 14, 46, 33, 36, 48, 72, 75, 50, 86`

In caso di errore (puntatore nullo, o indici non validi), la funzione non deve modificare il vettore fornito. Notate che to può essere uguale a len, perché la posizione to è esclusa.

Esercizio 4 (8 punti)

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

```
struct sparse_matrix {
    uint32_t rows, cols;
    uint32_t nnz;
    uint32_t *rowidxs;
    uint32_t *colidxs;
    double *data;
};
```

e la funzione:

```
extern double sm_get(struct sparse_matrix *m, uint32_t row, uint32_t col);
```

La struct consente di rappresentare, in forma sparsa, matrici di dimensioni arbitraria, dove `rows` è il numero di righe, `cols` è il numero di colonne. Le matrici sparse memorizzano solo i valori non nulli, quindi si utilizza il valore `nnz` per sapere quanti elementi non nulli ci sono e i vettori puntati da `rowidxs`, `colidxs` e `data` sono grandi `nnz` elementi e contengono rispettivamente per ogni elemento non nullo l'indice di riga, l'indice di colonna e il valore vero e proprio. Consideriamo ad esempio la matrice

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 9 \\ 0 & 0 & 5 & 0 \end{pmatrix}$$

Questa corrisponderebbe ad una variabile `struct sparse_matrix A`, con
`A.rows = 4`
`A.cols = 4`
`A.nnz = 3`
`A.rowidxs → { 1, 2, 3 }`
`A.colidxs → { 1, 3, 2 }`
`A.data → { 7.0, 9.0, 5.0 }`
Il simbolo → sta per "punta ad un area di memoria contenente".

La funzione accetta come parametri un puntatore ad una matrice sparsa `m` e una coordinata (`r`,`c`) e restituisce il valore dell'elemento della matrice a quella coordinata. Per fare questo deve cercare

se esiste un indice `i` per cui `rowidxs[i]` vale `r` e `colidxs[i]` vale `c`. In quel caso `data[i]` contiene il valore, altrimenti il valore è `0`.

Se il puntatore passato alla funzione è `NULL` o se r e c sono indici esterni alla matrice, la funzione ritorna `0`.

Esercizio 5 (7 punti)

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

```
extern void strip_comments(const char *in_filename, const char *out_filename);
```

La funzione accetta in input due stringhe C che contengono il nome di un file di input ed il nome di un file di output, da aprire rispettivamente in modalità lettura e scrittura tradotta (testo). La funzione deve leggere tutte le linee del file di input e copiarle in quello di output, tranne quelle che iniziano con il carattere #. Le linee possono avere lunghezza arbitraria.