Esercizio 1 (5 Punti)

Nel file `newton.c` inserire la definizione della funzione:

```
extern double newton(double m1, double m2, double d);
```

La funzione calcola la forza di attrazione --espressa in Newton-- di due corpi nell'universo, utilizzando la legge di gravitazione universale di Newton.

$$F = G\frac{m_1 \cdot m_2}{d^2}$$

La formula prevede che `m1` e `m2` siano la massa di due corpi celesti, espressa in kilogrammi, e che `d` sia la distanza tra essi espressa in metri. La costante di gravitazione universale `G` vale:

$$G = 6.67259 \times 10^{-11}$$

Se `m1` non è positivo, oppure se `m2` non è positivo, oppure se `d` non è positivo, la funzione restituisce `-1`.

Esercizio 2 (6 Punti)

Nel file `slicing.c` inserire la definizione della funzione:

```
extern int *slicing(const int *v, size_t *n, size_t start, size_t end,
                    int step);
```

Dato un vettore di numeri interi `v` e la sua lunghezza `n`, la funzione deve ritornare un nuovo vettore contenente i valori di `v` compresi nell'intervallo tra `start` ed `end` (inclusi) spostandosi di `step` elementi alla volta. `start` ed `end` vanno considerati partendo da zero, e saranno sempre minori di `n` e validi. **Attenzione**: se il parametro `step` è negativo, il verso di lettura dell'array `v` deve essere effettuato nella direzione inversa. La funzione aggiorna infine il valore di `n` inserendovi il numero degli elementi nel nuovo vettore.

Alcuni esempi:

```
Input:
    seq = {1, 2, 3, 4, 5, 6, 7, 8},
    start = 3
    end = 6
    step = 1

Output: {4, 5, 6, 7}
valore finale di n = 4
```

```
Input:
    seq = {1, 2, 3, 4, 5, 6, 7, 8},
    start = 2
```

```
    end = 6
    step = 2

Output: {3, 5, 7}
valore finale di n = 3
```

In questo caso gli elementi vengono copiati dall'indice 2 all'indice 6, ma selezionandone uno ogni due.

```
Input:
    seq = {1, 2, 3, 4, 5, 6, 7, 8},
    start = 2
    end = 6
    step = -1

Output: {7, 6, 5, 4, 3}
valore finale di n = 5
```

In questo caso lo step è negativo, quindi gli elementi dell'array vengono presi in direzione opposta.

```
Input:
    seq = {1, 2, 3, 4, 5, 6, 7, 8},
    start = 2
    end = 6
    step = -3

Output: {7, 4}
valore di n = 2
```

In questo caso oltre ad ordinare il vettore in senso opposto, viene selezionato solo un elemento ogni tre.

Esercizio 3 (7 Punti)

Nel file `hangman.c` inserire la definizione della funzione:

```c
extern char *hangman(const char* frase, const char* lettere);
```

La funzione accetta come parametri due stringhe C `frase` e `lettere`, e ritorna una nuova stringa C allocata dinamicamente su heap.

La funzione deve restituire una stringa C che sia una copia di `frase`, con il carattere `*` al posto delle lettere **non** presenti nella stringa `lettere`. Sono considerate lettere i 26 caratteri che formano l'alfabeto, maiuscoli e minuscoli. La stringa `lettere` contiene solo lettere minuscole. Per ogni lettera `c` inclusa in `lettere`, né le occorrenze di `c` né le occorrenze della sua versione maiuscola in `frase` devono essere sostituite da asterischi. Ad esempio, se `lettere` contiene il carattere `i`, non andranno sostituite né le occorrenze del carattere minuscolo `i`, né quelle del carattere maiuscolo `I`.

Se `frase` o `lettere` sono puntatori a NULL, la funzione restituisce NULL.

Alcuni esempi:

```
Input:
    frase = "Questa e' una frase di prova."
    lettere = "rsta"
Output: "***sta *' **a *ras* ** *r**a."
```

```
Input:
    frase = "Attenzione, la punteggiatura non e' mai nascosta!!"
    lettere = "no"
Output: "*******on*, ** **n********** non *' *** n*******!!"
```

```
Input:
    frase = "Il gioco dell'impiccato non e' case-sensitive"
    lettere = "aeiou"
Output: "I* *io*o *e**'i**i**a*o *o* e' *a*e-*e**i*i*e"
```

```
Input:
    frase = "Sopra la panca la capra campa, sotto la panca la capra crepa."
    lettere = ""
Output: "***** ** ***** ** ***** *****, ***** ** ***** ** ***** *****."
```

Esercizio 4 (7 Punti)

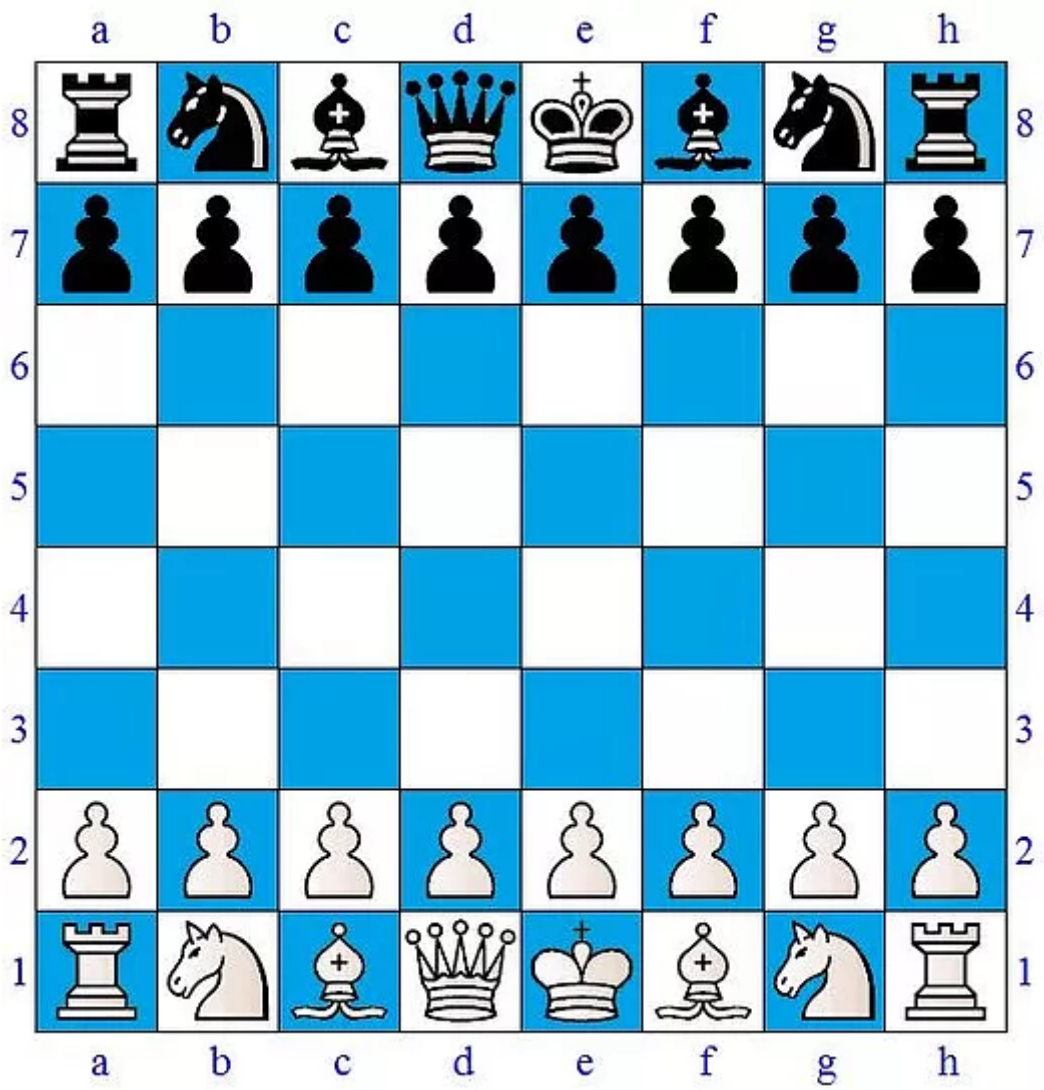
Creare il file `scacchiera.h` e `scacchiera.c` che consentano di utilizzare la seguente struttura:

```c
struct scacchiera {
    char caselle[64];
};
```

e la funzione:

```c
extern void stampa_scacchiera(const struct scacchiera *sc);
```

La struct `scacchiera` consente di rappresentare la posizione corrente dei pezzi su una scacchiera. L'array di 64 char `caselle` rappresenta una matrice 8x8 memorizzata per *colonne*: i primi 8 elementi di `caselle` corrispondono alla colonna a, dal basso verso l'alto, gli elementi dal 9 al 16 rappresentano la colonna b, e così via. Consideriamo ad esempio la posizione iniziale:



questo corrisponde ad una variabile struct `scacchiera s`, con `s.caselle` che che contiene 64 `char` ordinati per colonna, con la prima casella `s.caselle[0]` che corrisponde alla casella `a1` sulla scacchiera. Di conseguenza, il contenuto della casella`a2` sarà salvato in `s.caselle[1]`, e così di seguito fino ad `a8`, il cui contenuto è salvato in `s.caselle[7]`. Le colonne successive sono salvate con il medesimo ordine, fino a `h8` che corrisponde all'ultimo char dell'array `s.caselle[63]`. Le caselle vuote sono rappresentate con uno spazio, mentre ogni pezzo è rappresentato da una singola lettera.

La funzione `stampa_scacchiera` riceve in input una `struct scacchiera`, e deve stampare in output le 64 caselle della scacchiera come nell'esempio che segue, il quale contiene la posizione iniziale mostrata nella figura precedente.

```
+---+---+---+---+---+---+---+---+
| r | n | b | q | k | b | n | r |
+---+---+---+---+---+---+---+---+
| p | p | p | p | p | p | p | p |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
```

```
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
| P | P | P | P | P | P | P | P |
+---+---+---+---+---+---+---+---+
| R | N | B | Q | K | B | N | R |
+---+---+---+---+---+---+---+---+
```

Come mostrato nell'esempio, il contenuto di ogni casella è rappresentato da 3 caratteri: l'elemento dell'array `s` alla posizione corretta in mezzo a due spazi. Inoltre, ogni casella è delimitata dal carattere `+` ai suoi quattro angoli, dal singolo carattere `|` ai bordi orizzontali, e dai 3 caratteri `---` ai bordi verticali.

La `struct scacchiera` ricevuta in input sarà **sempre formattata correttamente**.

Esercizio 5 (8 Punti)

Un formato binario compatto per memorizzare valori reali nell'intervallo `[-2, 2)` con precisione circa alla quarta cifra decimale è ottenuto prendendo numeri a 16 bit in complemento a 2 e dividendoli per 2^{14}.

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

```c
struct dvec {
    size_t n;
    double *d;
};
```

e la funzione:

```c
struct dvec *read_dvec_comp(const char *filename);
```

La funzione apre il file `filename` in modalità non tradotta e legge dal file binario numeri a 16 bit in complemento a 2 in little endian, producendo in output una `struct dvec` allocata dinamicamente con `n` pari al numero di valori nel file, e `d` un puntatore ad un'area di memoria contenente gli `n` double corrispondenti ai valori codificati nel file. Se non è possibile aprire il file, o il file non contiene alcun valore, la funzione crea una `struct dvec` con `n=0` e `d=NULL`. Ad esempio il file: `00 80 CD AB FF FF 00 00 01 00 32 54 FF 7F` contiene i valori (rappresentati in base 10 con 6 cifre decimali): `-2.000000, -1.315613, -0.000061, 0.000000, 0.000061, 1.315552, 1.999939`