Esercizi matrici e strutture innestate

Si propongono alcuni esercizi relativi all'utilizzo di matrici (array bi-dimensionali) e strutture innestate (liste di array tramite puntatori di puntatori).

1. Implementare una funzione che, data una matrice come input, calcoli la somma di tutti i valori presenti nella sua "cornice". La funzione deve rispettare il seguente prototipo:

   ```
   long matrix_edge(unsigned n_rows, unsigned n_cols, int m[][n_cols]);
   ```

   dove:

   - `m` è la matrice di input, con `n_rows` righe e `n_cols` colonne;
   - la funzione calcola il risultato e restuisce la somma tramite il suo valore di ritorno.

2. Implementare una funzione che estrae la diagonale principale e la diagonale secondaria di una matrice quadrata. La funzione deve rispettare il seguente prototipo:

   ```
   void diagonals(int *rdp, int *rds, unsinged size, const int m[][size]);
   ```

   dove:

   - `m` è la matrice di input di dimensione `size` righe e `size` colonne;
   - `rdp` è il puntatore in cui viene memorizzata la diagonale principale;
   - `rds` è il puntatore in cui viene memorizzata la diagonale secondaria.
   - assumere che sia `rdp` sia `rds` siano puntatori ad aree di memoria già allocate correttamente.

3. Implementare una funzione che crei una matrice contenente la *tabula recta* (vedere wikipedia) dell'alfabeto inglese. Rispettare il seguente prototipo:

   ```
   void tabula_recta(char t[][26]);
   ```

   dove:

   - `t` è la matrice in cui viene salvato la matrice generata.

4. Implementare due funzioni utili per allocare e deallocare una lista di stringhe. Rispettare i seguenti prototipi:

   ```
   char **crea_lista(unsigned list_size, const unsigned *sizes);
   void distruggi_lista(char **list_p);
   ```

   dove:

   - `crea_lista` crea la lista allocando la memoria necessaria:
     - `list_size` è la dimensione della lista
     - `sizes` è il puntatore a un array di dimensione `list_size` che contiene le dimensioni da allocare per ogni elemento della lista;
     - la funzione restituisce il puntatore alla lista, e vale `NULL` in caso di errore di memoria. Fare in modo che la lista risulti inizializzata con stringhe *vuote*;
   - `distruggi_lista` dealloca la memoria della lista puntata dalla variabile `list_p` ricevuta in input.

5. Implementare una funzione che, data una stringa contenente una sequenza di parole separate da spazio, crei una lista di stringhe in cui l'ultimo elemento della lista è seguito da un valore NULL (ad esempio, in presenza di una stringa C di N parole, se la variabile `char **list` punta alla lista, `list[N]` è uguale a `NULL`). La funzione deve rispettare il seguente prototipo:

   ```
   char **split(const char *s);
   ```

   dove:

   - `s` è il puntatore alla stringa C data in input;
   - la funzione funzione restituisce il puntatore alla lista, e vale `NULL` in caso di errore di memoria.