Esercizi array e stringhe

I seguenti esercizi sono pensati per l'utilizzo di array e stringhe allocate automaticamente, senza utilizzare allocazione dinamica con `malloc`. Per questo si assuma che tutti i puntatori impiegati facciano già riferimento ad aree di memoria valida e di dimensioni appropriate.

1. Implementare una funzione che inverta l'ordine dei valori di un array di dati interi che rispetti il seguente prototipo:

   ```
   void reversei(int *r, const int *values, unsigned size);
   ```

   dove:

   - `r` è il puntatore all'array dove verrà salvata l'array invertito
   - `values` è il puntatore all'array di input
   - `size` è la dimensione dell'array

2. Implementare una funzione che inverta l'ordine dei caratteri di una stringa C che rispetti il seguente prototipo:

   ```
   void reverses(char *r, const char *s);
   ```

   dove:

   - `r` è il puntatore all'array dove verrà salvata la stringa C invertita
   - `s` è il puntatore alla stringa in input

3. Implementare una funzione `rot13` che trasorma una stringa in un'altra stringa di uguale dimensione, ma in cui ogni carattere della stringa generate è dal carattere alla stessa posizione nella stringa originale "spostato" di 13 caratteri (considerando una successione di caratteri "ciclica", dove la `a` segue la `z`).

   ```
   void rot13(char *r, const char *s);
   ```

   dove:

   - `r` è il puntatore alla stringa dove verrà salvata la stringa in output
   - `s` è il puntatore alla stringa in input

4. Implementare una funzione `findi` che individua la posizione di un valore all'interno di un array di interi. La funzione rispetti il seguente prototipo:

   ```
   long fini(int t, const int *values, unsigned size);
   ```

   dove:

   - `values` è il puntatore all'array di input
   - `size` è la dimensione dell'array
   - `t` è il valore da ricercare
   - il valore di ritorno indica la posizione all'interno dell'array, e ha valore speciale `-1` se l'elemento non esiste

5. Implementare una funzione `capitalize` che, data una stringa C di input, ne generi un'altra in cui tutte le lettere alfabetiche che seguono uno spazio siano maiuscole, mentre tutte le altre minuscole. Assumere che l'input possa avere sia lettere maiuscole sia minuscole sparse, e anche altri simboli.

   ```
   void capitalize(char *r, const char *s);
   ```

   dove:

- `r` è il puntatore alla stringa di output
- `s` è il puntatore alla stringa in input

6. Implementare una funzione per il calcolo delle frequenza di caratteri all'interno di una stringa. La funzione conta quante volte compare ogni carattere all'interno di una stringa e memorizza il risultato in un array:

   ```
   void freqs(unsigned *r, const char *s);
   ```

   dove:

   - `r` è il puntatore all'array generato con il risultato. Assumere che il puntatore si riferisca a un array di dimensioni 26, che possa contenere il conteggio di tutte le lettere dell'alfabeto inglese
   - `s` è il puntatore alla stringa di input

7. Implementare una funzione che dati due array di valori interi ordinati, generi un terzo array che contenga tutti i valori dei precedenti array in modo ordinato. Assumere che all'interno degli array ci possano essere elementi duplicati. La funzione rispetti il seguente prototipo:

   ```
   void merge(int *r, const int *a1, unsigned s1, const int *a2, unsigned s2);
   ```

   dove:

   - `r` è il puntatore all'array generato
   - `a1` è il puntatore al primo array di input
   - `s1` è la dimensione del primo array
   - `a2` è il puntatore al secondo array di input
   - `s2` è la dimensione del secondo array

8. Implementare una funzione per il calcolo della serie di Fibonacci, in cui si calcolano i primi `N` valori e li si salvano in un array. La funzione rispetti la seguente interfaccia:

   ```
   void fibonacci(unsigned *r, unsigned n);
   ```

   dove:

   - `r` è il puntatore all'array generato
   - `n` è il numero di valori della serie da generate