Esercizi strutture dati

Si propongono alcuni esercizi riguardo l'impiego di strutture dati, alcuni dei quali sono varianti di esercizi già affrontati in precedenza utilizzando tipi primitivi del C.

1. Implementatre una funziona per l'individuazione del valore minimo, del valore massimo, della media e della varianza dei valori di un array. La funzione deve rispettare il seguente prototipo:

   ```
   typedef struct array_info {
       int max;
       int min;
       float mean;
       float variance;
   } array_info_t;

   array_info_t array_stats(const int *values, unsigned size);
   ```

   dove:

   - `array_info_t` è la struct che gestisce la informazioni dell'array;
   - `values` è il puntatore all'array;
   - `size` è la dimensione dell'array.

2. Implementare delle varianti dell'esercizio precedente che implementino le stesse funzionalità con lo stesso tipo di struct, ma che supportino differenti prototipi.

   - *variante 1*:

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

     dove:

     - `r` è il puntatore alla struttura dati in cui memorizzare i valori calcolati dalla funzione.

   - *variante 2*:

     ```
     array_info_t *array_stats(const int *values, unsigned size);
     ```

     dove:

     - la funzione restituire il puntatore alla struttura dati allocata appositamente dalla funzione, in cui sono memorizzati i risultati. Restituisce `NULL` in caso di errore.

   - *variante 3*:

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

     dove:

     - `r` è il puntatore al puntatore della struttura dati allocata appositamente dalla funzione in cui sono memorizzati i risultati. Vale `NULL` in caso di errore.

3. 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:

   ```
   typedef struct int_array {
       int *p;
       unsigned size;
   } int_array_t;

   void merge(int_array_t **r, const int_array_t *a1, const int_array_t *a2);
   ```

dove:

- `int_array_t` è la struttura per gestire un array di variabili `int`. La struttura include la variabile puntatore `p`, che include l'indirizzo dell'array gestito, e variabile `size`, che indica la dimensione dell'array;
- `r` è il puntatore alla struct che gestisce l'array generato, da allocare appositamente da parte della funzione;
- `a1` è il puntatore alla struct che gestisce il primo array di input
- `a2` è il puntatore alla struct che gestisce il secondo array di input

4. Implementare una funzione per il calcolo dell'area di un poligono irregolare date le coordinate dei suoi verici. Si ricorda la formula per il calcolo dell'area $A = \frac{1}{2}|\sum_{i=1}^{n}[(x_i y_{i+1}) - (x_{i+1} y_i)]|$, dove x_{n+1} e y_{n+1} sono da intendere *modulo n*, ovvero x_1 e y_1. La funzione supporti il seguente prototipo:

```
typedef struct point {
    float x;
    float y;
} point_t;

typedef struct polygon {
    unsigned n;
    point_t *vertices;
} polygon_t;

float compute_area(const polygon_t *p);
```

dove:

- `point_t` rappresenta un punto sul piano con coordinate `x` e `y`;
- `polygon_t` rappresenta il poligono con `n` vertici, ognuno identificato come un punto sul piano;
- `compute_area` è la funzione per il calcolo dell'area:
  - accetta il puntatore `p` alla struttura che rappresenta il poligono;
  - restituisce l'area del poligono.

5. Implementare una funzione che, data una stringa C contenente una sequenza di parole separate da spazio, individui tutte le parole e le *indicizzi* in una struttura dati apposita che referenzi la stringa originale. La funzionalità è simile a quanto richiesto nell'ultimo esercizio della parte 5, ma si richiede esplicitamente di non creare una struttura che copi le singole stringhe presenti nella stringa originale. Si lascia libertà di scelta nell'implementazione della struttura necessaria, che verrà gestita da eventuali utilizzatori come struttura *opaca* e delle funzioni correlate, con il vincolo di rispettare il seguente prototipo. Si ricorda che è possibile implementare anche altre funzioni per gestire più facilmente la logica richiesta, che l'utilizzatore sfrutterà solo quelle indicate dai prototipi.

```
typedef struct string_index string_index_t;

string_index_t *index_string(const char *s);

void print_string(const string_index_t *st, unsigned i);

void destroy_index(string_index_t *st);
```

dove:

- `string_index_t` è la struttura opaca che gestisce l'indice delle stringhe;
- `index_string` accetta come input la stringa C `s` e restituisce il puntatore alla struct allocata all'interno dalla funzione stessa in memoria dinamica;
- `print_string` visualizza a monitor la `i`-esima parola della stringa utilizzando la struttura indice referenziata dal puntatore `st` (dove `i=0` indica la prima parola). Assumere che il valore di `i` sia sempre valido (ovvero, la frase iniziale abbia almeno `i-1` parole);
- `destroy_index` dealloca opportunamente la memoria impiegata nella struttura referenziata dal puntatore `st`.