Esercitazione alberi

1) Definire il tipo albero_s_t di un albero binario in cui le etichette sono stringhe di al piu' 20 caratteri. Scrivere una funzione

int conta_occorrenze (albero_s_t * root, char * s);

che conta le occorrenze della stringa s nell'albero di radice root e restituisce tale numero.

2) Definire il tipo albero_d_t di un albero binario in cui le etichette sono valori di tipo double. Scrivere una funzione

double somma_le_foglie (albero_d_t * root);

che restituisce la somma dei valori delle etichette delle sole foglie dell'albero.

3)Utilizzando il tipo albero_d_t dell'esercizio precedente. Scrivere una procedura

void leggi_albero (albero_d_t ** root) ;

che legge da standard input una sequenza di valori double terminata da EOF e restituisce il puntatore ad un nuovo albero che contiene tutti i valori letti. L'albero deve essere costruito in modo da essere ordinato, cioe' dato un nodo n tutti i valori nel sottoalbero sinistro devono essere minori o uguali del valore in n, mentre i valori nel sottoalbero destro devono essere tutti maggiori o uguali di n.

Verificare che stampando i valori delle etichette con una visita simmetrica otteniamo una sequenza crescente.

4)Utilizzando il tipo `albero_d_t` dell'esercizio precedente, scrivere una funzione che trasforma l'albero in una lista di double inserendo le etichette nell'ordine della visita anticipata

```
lista_d_t * tree_to_list ( albero_d_t * root );
```

5) Dato il tipo albero_d_t dell'esercizio precedente, scrivere una procedura che stampa la visita a livelli dell'albero. Suggerimento: utilizzare un array per memorizzarsi i puntatori ai figli...

6) Un albero binario di ricerca e' un albero in cui in ogni nodo *n* e' verificata la relazione

$$E(nsx) \leq E(n) \leq E(ndx)$$

dove *nsx* e' un qualsiasi nodo dell'albero di sinistra e *ndx* e' un qualsiasi nodo dell'albero di destra. Utilizzando il tipo albero_d_t realizzate le seguenti funzioni/procedure (a scelta):

/* inserisce l'etichetta x nell'albero mantenedolo ordinato, restituisce il puntatore al nuovo albero */

..... inserisci_ord (....., double x);

/* ricerca l'etichetta x nell'albero analizzando un numero di nodi pari all'altezza dell'albero
 restituisce 1 se la trova e 0 se non la trova */

```
int inserisci_ord ( albero_d_t * root, double x );

/* cancella l'etichetta x nell'albero mantenedolo ordinato e deallocando l'etichetta (difficile!)
restituice il puntatore al nuovo albero */
..... cancella_ord ( ....., double x );
```