```
/*        Quimera, luglio 2003          *
 *           Versione 1.0.0             *
 *                                      *
 * Libreria per la gestione di oggetti *
 * all'interno di un albero binario.    *
 *                                      *
 * Dichiarazione dei tipi di variabili *
 * e di procedure utili alla libreria. */

#ifndef _TREE_H
#define _TREE_H

/* ---------------------- Dichiarazione delle variabili ----------------------- */

struct NODO_T {                /* Struttura del nodo dell'albero */
  void *elem;
  struct NODO_T *ns, *nd;
};
typedef struct NODO_T nodo_t;

typedef nodo_t *tree_t;        /* Albero                         */

/* ------------------ Dichiarazione delle funzioni e procedure ---------------- */

tree_t new_tree (void);
/* Requisiti : nessuno                                  *
 * Ruolo: crea un albero.                               */

tree_t right_son (tree_t);
/* Requisiti : albero inizializzato e non vuoto         *
 * Ruolo: Restituisce il figlio destro dell'albero,     *
 *   NULL altrimenti.                                   */

tree_t left_son (tree_t);
/* Requisiti : albero inizializzato e non vuoto         *
 * Ruolo: Restituisce il figlio sinistro dell'albero,   *
 *   NULL altrimenti.                                   */

void * min_of_tree (tree_t);
/* Requisiti : albero inizializzato e non vuoto         *
 * Ruolo: Restituisce l'elemento più piccolo dell'albero  */

void * max_of_tree (tree_t);
/* Requisiti : albero inizializzato e non vuoto         *
 * Ruolo: Restituisce l'elemento più grande dell'albero   */

int tree_is_empty (tree_t);
/* Requisiti : albero inizializzato                     *
 * Ruolo: Restituisce 1 se l'albero è vuoto,            *
 *   0 altrimenti.                                      */

void ** get_elem (tree_t);
/* Requisiti : albero inizializzato e non vuoto         *
 * Ruolo: Restituisce il puntatore sulll'elemento della *
 *   radice dell'albero                                 */

int add_elem (tree_t *, void *, int (*) (void *, void *));
/* Requisiti : albero inizializzato                     *
 * Ruolo: aggiunge all'albero l'elemento se non è       *
 *   presente. Se l'operazione ha avuto successo        *
 *   restituisce 1 altrimenti 0. Per fare cio, ha bisogno *
 *   di una funzione di comparazione per gli elementi;  *
 *   questa rende -1, 0, 1 a seconda se il 1 oggetto è  *
 *   rispetivamente minore, uguale o maggiore del secondo */
```



```
int rem_elem (tree_t *, void *, int (*) (void *, void *));
/* Requisiti : albero inizializzato                     *
 * Ruolo: aggiunge all'albero l'elemento, se            *
 *   l'operazione ha successo restituisce 1 altrimenti 0. *
 *   Per fare cio, ha bisogno di una funzione di        *
 *   comparazione per gli elementi; questa rende -1, 0, 1 *
 *   a seconda se il 1 oggetto è rispetivamente minore, *
 *   uguale o maggiore del secondo                      */

void ** find_elem (tree_t, void *, int (*) (void *, void *));
/* Requisiti : albero inizializzato e non vuoto         *
 * Ruolo: cerca all'interno dell'albero l'elemento, se lo *
 *   trova restituisce l'elemento, altrimenti restituisce *
 *   NULL. Per fare cio, ha bisogno di una funzione di  *
 *   comparazione per gli elementi; questa rende -1, 0, 1 *
 *   a seconda se il 1 oggetto è rispetivamente minore, *
 *   uguale o maggiore del secondo                      */

#endif
```

```
/*        Quimera, luglio 2003          *
 *           Versione 1.0.0             *
 *                                      *
 * Libreria per la gestione di oggetti *
 * all'interno di un albero binario.    *
 *                                      *
 * Definizione delle variabili e delle *
 * procedure utili alla libreria.       */

#include "tree.h"
#include <stdlib.h>
#include <stdio.h>

/* ---------------------------------- new_tree ---------------------------------- */

tree_t new_tree (void) {

  return NULL;
}

/* ---------------------------------- right_son ---------------------------------- */

tree_t right_son (tree_t tree) {

  return (*tree).nd;
}

/* ---------------------------------- left_son ---------------------------------- */

tree_t left_son (tree_t tree) {

  return (*tree).ns;
}

/* ---------------------------------- tree_is_empty ---------------------------------- */

int tree_is_empty (tree_t tree) {

  if ( tree == NULL )
    return 1;
  else
    return 0;
}

/* ---------------------------------- get_elem ---------------------------------- */

void ** get_elem (tree_t tree) {

  return &(*tree).elem;
}

/* ---------------------------------- min_of_tree ---------------------------------- */

void * min_of_tree (tree_t tree) {

  if ( tree_is_empty(left_son(tree)) )
    return *(get_elem(tree));
  else
    return min_of_tree(left_son(tree));
}

/* ---------------------------------- min_of_tree ---------------------------------- */

void * max_of_tree (tree_t tree) {
```



```
  if ( tree_is_empty(right_son(tree)) )
    return *(get_elem(tree));
  else
    return max_of_tree(right_son(tree));
}


/* ---------------------------------- add_elem ---------------------------------- */

int add_elem (tree_t *tree, void *elem, int (* f) (void *, void *)) {

  nodo_t *nodo;

  if ( tree_is_empty(*tree) )
    if ( (nodo = malloc(sizeof(nodo_t))) == NULL )
      return 0; /* problema nell'allocazoine della memoria */
    else { /* inizializzazione del nodo */
      (*nodo).elem = elem;
      (*nodo).ns = NULL;
      (*nodo).nd = NULL;
      *tree = nodo;
      return 1;
    }
  else { /* l'albero non è vuoto */
    /* inserire nella giusta posizione l'elemento */
    switch ( f(elem, (**tree).elem) ) {
    case -1 :
      return add_elem(&(**tree).ns, elem, f);
      break;
    case 0 :
      return 0;
      break;
    case 1 :
      return add_elem(&(**tree).nd, elem, f);
      break;
    default :
      return 0;
    }
  }
}

/* ---------------------------------- rem_elem ---------------------------------- */

int rem_elem (tree_t *tree, void *key, int (* f) (void *, void *)) {

  tree_t tmp;

  if ( tree_is_empty(*tree) )
    /* l'albero è vuoto */
    return 0;

  if ( tree_is_empty(left_son(*tree)) ) {
    /* l'albero non ha figli sinistri */
    if ( !f((**tree).elem, key) ) {
      tmp = *tree;
      *tree = right_son(*tree);;
      free(tmp);
      return 1;
    }
    else
      /* ricorsività sul lato destro dell'albero */
      return rem_elem(&(**tree).nd, key, f);
  }
```

```
  if ( tree_is_empty(right_son(*tree)) ) {
    /* l'albero non ha figli destri */
    if ( !f((**tree).elem, key) ) {
      tmp = *tree;
      *tree = left_son(*tree);
      free(tmp);
      return 1;
    }
    else
      /* ricorsività sul lato sinistro dell'albero */
      return rem_elem(&(**tree).ns, key, f);
  }

  /* albero ha entrambi i figli */
  if ( !f((**tree).elem, key) ) {
    (**tree).elem = max_of_tree(left_son(*tree));
    return rem_elem(&(**tree).ns, (**tree).elem, f);
  }
  else
    /* ricorsività sul lato giusto dell'albero */
    switch ( f(key, (**tree).elem) ) {
    case -1 :
      return rem_elem(&(**tree).ns, key, f);
      break;
    case 0 :
      return 0;
      break;
    case 1 :
      return rem_elem(&(**tree).nd, key, f);
      break;
    default :
      return 0;
    }
}

/* ------------------------------ find_elem ------------------------------ */

void ** find_elem (tree_t tree, void *key, int (* f) (void *, void *)) {

  if ( tree_is_empty(tree) )
     return NULL;

  switch ( f(key, *(get_elem(tree))) ) {
  case -1 :
    return find_elem (left_son(tree), key, f);
    break;
  case 0 :
    return get_elem(tree);
    break;
  case 1 :
    return find_elem (right_son(tree), key, f);
    break;
  default :
    return NULL;
  }
}
```

```
#include "tree.h"

int maggiore (void *a, void *b) {

  if ( a > b )
    return 1;
  else
    if ( a == b )
      return 0;
    else
      return -1;
}

int main (void) {

  tree_t tree = new_tree();
  int i, res;

  i = 12;
  add_elem(&tree, (void *)i, maggiore);
  i = 15;
  add_elem(&tree, (void *)i, maggiore);
  i = 5;
  add_elem(&tree, (void *)i, maggiore);
  i = 7;
  add_elem(&tree, (void *)i, maggiore);
  i = 3;
  add_elem(&tree, (void *)i, maggiore);
  i = 34;
  add_elem(&tree, (void *)i, maggiore);
  i = 28;
  add_elem(&tree, (void *)i, maggiore);
  i = 1;
  add_elem(&tree, (void *)i, maggiore);
  i = 10;
  add_elem(&tree, (void *)i, maggiore);
  i = 14;
  add_elem(&tree, (void *)i, maggiore);

  print_tree(tree, " %d ");

  i = 5;
  rem_elem(&tree, (void *)i, maggiore);

  print_tree(tree, " %d ");

  i = 10;
  res = (int)*find_elem(tree, (void *)i, maggiore);

  printf("---%d---\n", res);

  return 0;
}
```