CI2126 QUIZ 2 5%

Dado el código de la página siguiente que representa un TAD Arbol (Binario), implemente la operación

a) **Arbol** *eliminarArbolElem(**Arbol*** *a,* ***Elem*** *e)*

Ejemplo: Se recibe este árbol de la forma **(info hizq hder)**

```
( 12 ( 3 1 ( 8 5 10 ) ) ( 19 15 23 ) )
```

Se eliminan en orden los nodos, resultando en los árboles

```
 5, --> ( 12 ( 3 1 ( 8 · 10 ) ) ( 19 15 23 ) )
 8, --> ( 12 ( 3 1 10 ) ( 19 15 23 ) )
15, --> ( 12 ( 3 1 10 ) ( 19 · 23 ) )
12, --> ( 10 ( 3 1 · ) ( 19 · 23 ) )
```

- Si es hoja, eliminar hoja y ajustar el padre. Eliminar Nodo
- Si es nodo con un solo hijo, subir el hijo, ajustar el padre subiendo este hijo. Eliminar nodo
- Si es nodo con dos hijos, reemplazar el valor del nodo con mayor valor en el subarbol izquierdo y eliminar el nodo con el mayor valor en el subarbol izquierdo.

En Rojo: No son parte de la respuesta, se coloca a modo ilustración.
En Verde: Código a remover si no se usara el apuntador al padre.

```c
#include <stdio.h>
#include <stdlib.h>
#include "hoare.h"

typedef struct ArbolCDT_t* Arbol;
typedef struct ArbolCDT_t {
    Elem  info;   /* Elem es "Int"    */
    Arbol hizq;   /* Hijo izquierdo   */
    Arbol hder;   /* Hijo derecho     */
    Arbol padre;  /* Puntero al padre */
} ArbolCDT_t;

Arbol consHoja(Elem e) {   /* Crea nodo de árbol sin hijos ni padre */
    Arbol a =  CREATE(ArbolCDT_t);
    a->info =  e;
    a->hizq = a->hder = a->padre = NULL;
    return a;
}

Arbol consArbol(Elem e, Arbol hi, Arbol hd, Arbol pa) {
    Arbol a =  consHoja(e);
    a->hizq  = hi;   /* Asignar subarbol hijo izquierdo   */
    a->hder  = hd;   /* Asignar subarbol hijo derecho     */
    a->padre = pa;   /* Asignar puntero al padre          */
    return a;
}

bool esHoja(Arbol a) {
    return (a and !a->hizq and !a->hder);
}

bool esRaiz(Arbol a) {
    return (a and !a->padre);
}

    /* Inserta ordenado en el arbol */
Arbol insertarArbolElem(Arbol* a, Elem e);
```

```
    /* Busca ordenado el mayor valor del arbol por el subarbol izquierdo (sólo por los hijos derechos) */
Arbol buscarMayorIzq(Arbol a);

int main () {
    Arbol a;
    insertarArbolElem(&a, 12); // etc
    ...
    eliminarArbolElem(&a,  5);
    eliminarArbolElem(&a,  8);
    eliminarArbolElem(&a, 15);
    eliminarArbolElem(&a, 12);

    imprimirArbol(a);
    return 0;
}
```

SOLUCIÓN

```
     /* Devuelve   -1: a<b,   0: a==b,   1: a>b  */
inline int compararElem (Elem a, Elem b) {
    _pre(a and b);
    return ( a->item - b->item );
}

    /* Busca ordenado el mayor valor del arbol por el subarbol izquierdo (sólo por los hijos derechos) */
Arbol buscarMayorIzq(Arbol a) {
    _pre(a);
    if (a->hder)
        return buscarMayorIzq(a->hder]);
    return a;
}

    /* Elimina el Elem e del Arbol a */
Arbol eliminarArbolElem(Arbol* a, Elem e) {
    if (*a) {
        if      (compararElem(e, (*a)->info) < 0)     // e < (*a)->info, eliminamos por el subarbol izquierdo
            return eliminarArbolElem(&(*a)->hizq, e);
        else if (compararElem(e, (*a)->info) > 0)     // e > (*a)->info, eliminamos por el subarbol derecho
            return eliminarArbolElem(&(*a)->hder, e);
            /* Lo encontramos, e == (*a)->info,  compararElem(e, (*a)->info) == 0 */
        if (esHoja(*a)) {                             // Caso 1: es una hoja
            destNodoArbol(a);                         //    Eliminamos la hoja
        } else if (!(*a)->hizq) {                     // Caso 2d: No hay hijo izquierdo, subimos hijo derecho
            Arbol r = *a;                             // Copiamos apuntador al nodo
            r->hder->padre = (*a)->padre;             // Ajustamos el apuntador al padre
            *a = (*a)->hder;                          // subimos hijo derecho
            destNodoArbol(&r);                        //    Eliminamos el nodo
        } else if (!(*a)->hder) {                     // Caso 2i: No hay hijo derecho, subimos hijo izquierdo
            Arbol r = *a;                             // Copiamos apuntador al nodo
            r->hizq->padre = (*a)->padre;             // Ajustamos el apuntador del padre
            *a = (*a)->hizq;                          // subimos hijo izquierdo
            destNodoArbol(&r);                        //    Eliminamos el nodo
        } else {                                      // Caso 3: Ambos subárboles existen,
            Arbol t = buscarMayorIzq((*a)->hizq);     // Buscamos por mayor valor del subarbol izquierdo
            _pre(t);                                  // t es el nodo con mayor valor del subarbol izquierdo
            SWAP((*a)->info, t->info);                // intercambiamos info de los nodos
            if (t == (*a)->hizq)                      // Caso especial: Si además t es hijo izq de la raiz
                (*a)->hizq = NULL;                    //    ajustamos hijo izq de la raiz
            else                                      // sino,
                t->padre->hder = NULL;                //    lo desconectamos de su padre
            eliminarArbolElem(&t, t->info);           // Eliminamos recursivamente el nodo t cambiado
        }
    }
    return *a;                                        // Devolvemos el valor de *a por si acaso
}
```