

Examen Febrero

Implementa una función genérica que transforme un árbol binario de un tipo genérico T, eliminando los descendientes propios de todo aquellos nodos cuyo contenido sea, al mismo tiempo, mayor o igual que el de sus ascendientes propios y menor o igual que el de sus descendientes propios.

```
#include <iostream>
#include "../abin.h"
#include "../abin_E-S.h"

template <typename T> void
elimina_nodo(Abin<T>&A){
    if(!A.arbolVacio()){
        elimina_nodo_rec(A.raiz(),A);
    }
}

template <typename T> bool
menor_descendiente(typename Abin<T>::nodo n, const Abin<T>& A){
    if(n==Abin<T>::NODO_NULO) return true;
    else return (A.elemento(n)<= A.elemento(A.hijoIzqdo(n)) &&
        A.elemento(n) <=A.elemento(A.hijoDrcho(n)));
}

template <typename T> bool
mayor_ascendiente(typename Abin<T>::nodo n,const Abin<T>&A){
    if(n==Abin<T>::NODO_NULO) return true;
    else return (A.elemento(n) >=A.elemento(A.padre(n)));
}

template <typename T> bool
EsHoja(typename Abin<T>::nodo n, const Abin<T>&A){
    return (A.hijoIzqdo(n) == Abin<T>::NODO_NULO &&
        A.hijoDrcho(n)==Abin<T>::NODO_NULO);
}

template <typename T> void
hundir_nodo(typename Abin<T>::nodo n, Abin<T>&A){
    if(n!=Abin<T>::NODO_NULO){
        if(A.hijoIzqdo(n)!=Abin<T>::NODO_NULO){
            T aux = A.elemento(n);
            A.elemento(n) = A.elemento(A.hijoIzqdo(n));
            A.elemento(A.hijoIzqdo(n)) = aux;
            hundir_nodo(A.hijoIzqdo(n),A);
        }
        else if(A.hijoDrcho(n)!=Abin<T>::NODO_NULO){
            T aux = A.elemento(n);
            A.elemento(n) = A.elemento(A.hijoDrcho(n));
            A.elemento(A.hijoDrcho(n)) = aux;
            hundir_nodo(A.hijoDrcho(n),A);
        }
    }
    else{

```

```

        if (A.hijoIzqdo(A.padre(n)) == n)
            A.eliminarHijoIzqdo(A.padre(n));
        else
            A.eliminarHijoDrcho(A.padre(n));
    }
}

template <typename T> void
elimina_nodo_rec(const typename Abin<T>::nodo n, Abin<T>& A){
    if(n!=Abin<T>::NODO_NULO){
        if(A.hijoIzqdo(n)!=Abin<T>::NODO_NULO && A.hijoDrcho(n)!=Abin<T>::NODO_NULO){
            if(menor_descendiente(n,A) && mayor_ascendiente(n,A)){
                if(!EsHoja(A.hijoIzqdo(n),A)){
                    hundir_nodo(A.hijoIzqdo(n),A);
                }else A.eliminarHijoIzqdo(n);
            }else if(!EsHoja(A.hijoDrcho(n),A)){
                hundir_nodo(A.hijoDrcho(n),A);
            }else A.eliminarHijoDrcho(n);
        }
        elimina_nodo_rec(A.hijoIzqdo(n),A);
        elimina_nodo_rec(A.hijoDrcho(n),A);
    }
}

```

• Programa de prueba

```

#include "EliminaDescendientes.hpp"
#include <iomanip>
int main() {
    // Crear un árbol binario con algunos elementos
    Abin<int> A;
    A.insertarRaiz(10);
    A.insertarHijoIzqdo(A.raiz(), 50);
    A.insertarHijoDrcho(A.raiz(), 1);
    A.insertarHijoIzqdo(A.hijoIzqdo(A.raiz()), 110);
    A.insertarHijoDrcho(A.hijoIzqdo(A.raiz()), 60);
    A.insertarHijoDrcho(A.hijoDrcho(A.raiz()), 2);
    A.insertarHijoIzqdo(A.hijoDrcho(A.hijoDrcho(A.raiz())), 4);
    A.insertarHijoDrcho(A.hijoDrcho(A.hijoDrcho(A.raiz())), 3);
    A.insertarHijoIzqdo(A.hijoIzqdo(A.hijoIzqdo(A.raiz())), 70);
    A.insertarHijoDrcho(A.hijoIzqdo(A.hijoIzqdo(A.raiz())), 150);
    // Imprimir el árbol original
    imprimirAbin(A);
    // Eliminar nodos según las condiciones especificadas
    elimina_nodo(A);
    // Imprimir el árbol después de eliminar los nodos
    std::cout<<std::setw(20)<<std::setfill('=')<< ' ' <<"Eliminamos los nodos que no cumplen las condiciones especificadas" <<std::endl;
    imprimirAbin(A);
    return 0;
}

```