```
#ifndef Arbol_H_INCLUDED
#define Arbol_H_INCLUDED

#include <string>
#include <iostream>

///DEFINICION
template <class T>
class Arbol {
    private:
        class Node {
            private:
                T* dataPtr;
                Node* left;
                Node* right;

            public:
                Node();
                Node(const T&);

                ~Node();

                T* getDataPtr() const;
                T getData() const;
                Node*& getLeft();
                Node*& getRight();

                void setDataPtr(T*);
                void setData(const T&);
                void setLeft(Node*&);
                void setRight(Node*&);
            };

    public:
        typedef Node* Position;

    private:
        Position root;

        void copyAll(Arbol<T>&);

        void insertData(Position&, const T&);

        Position& findData(Position&, const T&);

        Position& getLowest(Position&);
        Position& getHighest(Position&);

        void parsePreOrder(Position&);
        void parseInOrder(Position&);
        void parsePostOrder(Position&);

        void deleteAll(Position&);

    public:
        Arbol();
        Arbol(Arbol<T>&);

        ~Arbol();

        bool isEmpty() const;

        void insertData(const T&);

        void deleteData(Position&);
```

```cpp
        T retrieve(Position&) const;

        int getHeight(Position&);

        int getLeftHeight();
        int getRightHeight();

        Position& findData(const T&);

        Position& getLowest();
        Position& getHighest();

        bool isLeaf(Position&) const;

        int getHeight();

        void parsePreOrder();
        void parseInOrder();
        void parsePostOrder();

        void deleteAll();
        void copyAll(Position&);

        Arbol& operator = (const Arbol&);
    };

///IMPLEMENTACION

///NODE
template <class T>
Arbol<T>::Node::Node() : dataPtr(nullptr), left(nullptr), right(nullptr) { }

template <class T>
Arbol<T>::Node::Node(const T& e) : dataPtr(new T(e)), left(nullptr), right(nullptr) {
    if(dataPtr == nullptr) {
        std::cout << "Memoria insuficiente, Node";
        }
    }

template <class T>
Arbol<T>::Node::~Node() {
    delete dataPtr;
    }

template <class T>
T* Arbol<T>::Node::getDataPtr() const {
    return dataPtr;
    }

template <class T>
T Arbol<T>::Node::getData() const {
    if(dataPtr == nullptr) {
        std::cout << "Dato inexistente, getData";        }

    return *dataPtr;
    }

template <class T>
typename Arbol<T>::Node*& Arbol<T>::Node::getLeft() {
    return left;
    }

template <class T>
typename Arbol<T>::Node*& Arbol<T>::Node::getRight() {
    return right;
    }
```

```
template <class T>
void Arbol<T>::Node::setDataPtr(T* p) {
    dataPtr = p;
    }

template <class T>
void Arbol<T>::Node::setData(const T& e) {
    if(dataPtr == nullptr) {
        if((dataPtr = new T(e)) == nullptr) {
            std::cout << ("Memoria no disponible, setData");
            }
        }
    else {
        *dataPtr = e;
        }
    }

template <class T>
void Arbol<T>::Node::setLeft(Node*& p) {
    left = p;
    }

template <class T>
void Arbol<T>::Node::setRight(Node*& p) {
    right =  p;
    }

///Arbol

template <class T>
void Arbol<T>::copyAll(Arbol<T>& bt) {
    copyAll(bt.root);
    }

template <class T>
void Arbol<T>::copyAll(Position& r) {
    if(r == nullptr) {
        return;
        }

    insertData(r->getData());
    copyAll(r->getLeft());
    copyAll(r->getRight());
    }

template <class T>
Arbol<T>::Arbol() : root(nullptr) { }

template <class T>
Arbol<T>::Arbol(Arbol<T>& t) : root(nullptr) {
    copyAll(t);
    }

template <class T>
Arbol<T>::~Arbol() {
    deleteAll();
    }

template <class T>
bool Arbol<T>::isEmpty() const {
    return root == nullptr;
    }

template <class T>
void Arbol<T>::insertData(const T& e) {
    insertData(root, e);
    }

template <class T>
void Arbol<T>::insertData(Position& r, const T& e) {
    if(r == nullptr) {
            if((r = new Node(e)) == nullptr) {
                std::cout << "Memoria no disponible, insertData";
                }
            }
    else {
        if(e < r->getData()) {
            insertData(r->getLeft(), e);
            }
        else {
            insertData(r->getRight(), e);
            }
        }
    }

template <class T>
void Arbol<T>::deleteData(Position& r) {
    if ( root == nullptr or r == nullptr ) {
        std::cout << "Posicion invalida, deleteData";
        }

    if ( isLeaf(r) ) {
        delete r;
        r = nullptr;
        }
    else {
        std::cerr << "\nasd\n";
        Position replacementPos;
        if ( r->getLeft() != nullptr ) {
            replacementPos = getHighest(r->getLeft());
            if (r->getLeft() != replacementPos) {
                r->setLeft(r->getLeft()->getLeft());
                }
            else {
                r->setLeft(r->getLeft());
                }
            }
        else {
            // cerr << "correct";
            replacementPos = getLowest(r->getRight());
            if (r->getRight() == replacementPos) {
                r->setRight(r->getRight()->getRight());
                }
            else {
                r->setRight(r->getRight());
                }
            }

        r->setData(replacementPos->getData());
        delete replacementPos;
        replacementPos = nullptr;
        }
    }

template <class T>
T Arbol<T>::retrieve(Position& r) const {
    return r->getData();
    }

template <class T>
typename Arbol<T>::Position& Arbol<T>::findData(const T& e) {
    return findData(root, e);
```

```cpp
    }

template <class T>
typename Arbol<T>::Position& Arbol<T>::findData(Position& r, const T& e) {
    if(r == nullptr or r->getData() == e) {
        return r;
        }

    if(e < r->getData()) {
        return findData(r->getLeft(), e);
        }

    return findData(r->getRight(), e);
    }

template <class T>
typename Arbol<T>::Position& Arbol<T>::getLowest() {
    return getLowest(root);
    }

template <class T>
typename Arbol<T>::Position& Arbol<T>::getLowest(Position& r) {
    if(r == nullptr or r->getLeft() == nullptr) {
        return r;
        }

    return getLowest(r->getLeft());
    }

template <class T>
typename Arbol<T>::Position& Arbol<T>::getHighest() {
    return getHighest(root);
    }

template <class T>
typename Arbol<T>::Position& Arbol<T>::getHighest(Position& r) {
    if(r == nullptr or r->getRight() == nullptr) {
        return r;
        }

    return getHighest(r->getRight());
    }

template <class T>
bool Arbol<T>::isLeaf(Position& r) const {
    return r != nullptr and r->getLeft() == r->getRight();
    }

template <class T>
int Arbol<T>::getHeight() {
    return getHeight(root);
    }

template <class T>
int Arbol<T>::getLeftHeight() {
    return getHeight(root->getLeft());
    }


template <class T>
int Arbol<T>::getRightHeight() {
    return getHeight(root->getRight());
    }

template <class T>
int Arbol<T>::getHeight(Position& r) {
    if(r == nullptr) {
        return 0;
        }

    int lH(getHeight(r->getLeft()));
    int rH(getHeight(r->getRight()));

    return (lH > rH ? lH : rH) + 1;
    }

template <class T>
void Arbol<T>::parsePreOrder() {
    parsePreOrder(root);
    }

template <class T>
void Arbol<T>::parsePreOrder(Position& r) {
    if(r == nullptr) {
        return;
        }

    std::cout << r->getData() << ", ";
    parsePreOrder(r->getLeft());
    parsePreOrder(r->getRight());
    }

template <class T>
void Arbol<T>::parseInOrder() {
    parseInOrder(root);
    }

template <class T>
void Arbol<T>::parseInOrder(Position& r) {
    if(r == nullptr) {
        return;
        }

    parseInOrder(r->getLeft());
    std::cout << r->getData() << ", ";
    parseInOrder(r->getRight());
    }

template <class T>
void Arbol<T>::parsePostOrder() {
    parsePostOrder(root);
    }

template <class T>
void Arbol<T>::parsePostOrder(Position& r) {
    if(r == nullptr) {
        return;
        }

    parsePostOrder(r->getLeft());
    parsePostOrder(r->getRight());
    std::cout << r->getData() << ", ";
    }

template <class T>
void Arbol<T>::deleteAll() {
    deleteAll(root);
    }

template <class T>
void Arbol<T>::deleteAll(Position& r) {
    if(r == nullptr) { ///CRITERIO DE PARO
```

```
        return;
        }

    deleteAll(r->getLeft());
    deleteAll(r->getRight());
    delete r;

    r = nullptr;
    }

template <class T>
Arbol<T>& Arbol<T>::operator=(const Arbol& t) {
    deleteAll();

    copyAll(t);

    return *this;
    }

#endif // Arbol_H_INCLUDED
```