```
#pragma once

#include <stdexcept>
#include <algorithm>

template<typename T>
struct BinaryTreeNode
{
    using BNode = BinaryTreeNode<T>;
    using BTreeNode = BNode*;

    T key;
    BTreeNode left;
    BTreeNode right;

    static BNode NIL;

    const T& findMax() const
    {
        if (empty())
        {
            throw std::domain_error("Empty tree encountered.");
        }

        return right ->empty() ? key : right ->findMax();
    }

    const T& findMin() const
    {
        if (empty())
        {
            throw std::domain_error("Empty tree encountered.");
        }

        return left ->empty() ? key : left ->findMin();
    }

    bool remove(const T& aKey, BTreeNode aParent)
    {
        BTreeNode x = this;
        BTreeNode y = aParent;

        while (!x->empty())
        {
            if (aKey == x->key)
            {
                break;
            }

            y = x;

            x = aKey < x->key ? x->left : x->right;
        }
```

```
        return true;
    }

        BinaryTreeNode() :key(T()), left(&NIL), right(&NIL)
        {

        }

        BinaryTreeNode(const T & aKey) :key(aKey), left(&NIL), right(&NIL)
        {

        }

        BinaryTreeNode(T && aKey) :key(std::move(aKey)), left(&NIL), right
          (&NIL)
        {

        }

        ~BinaryTreeNode()
        {
            if (!left->empty())
                delete left;

            if (!right->empty())
                delete right;
        }

        bool empty() const
        {
            return this == &NIL;
        }

        bool leaf() const
        {
            return left->empty() && right->empty();
        }

        size_t height() const
        {
            if (empty())
                throw std::domain_error("Empty Tree encountered");

            if (leaf())
                return 0;

            const size_t lLeftHeight = left->empty() ? 1 : left->height() +
              1;
            const size_t lRightHeight = right->empty() ? 1 : right->height()
               + 1;

            return std::max(lLeftHeight, lRightHeight);
```

```
        }

        bool insert(const T & aKey)
        {
            if (empty())
                return false;

            if (aKey > key)
            {
                if (right->empty())
                    right = new BNode(aKey);

                else return right->insert(aKey);

                return true;
            }

            if (aKey < key)
            {
                if (left->empty())
                    left = new BNode(aKey);

                else return left->insert(aKey);

                return true;
            }

            return false;
        }
};

template<typename T>
BinaryTreeNode<T> BinaryTreeNode<T>::NIL;
```