```cpp

// COS30008, Final Exam

#pragma once

#include "TernaryTree.h"

#include <stack>

template<typename T>
class TernaryTreePrefixIterator
{
private:
    using TTree = TernaryTree<T>;
    using TTreeNode = TTree*;
    using TTreeStack = std::stack<const TTree*>;

    const TTree* fTTree;                    // ternary tree
    TTreeStack fStack;                      // traversal stack

public:

    using Iterator = TernaryTreePrefixIterator<T>;

    Iterator operator++(int)
    {
        Iterator old = *this;

        ++(*this);

        return old;
    }

    bool operator!=( const Iterator& aOtherIter ) const
    {
        return !(*this == aOtherIter);
    }

///////////////////////////////////////////////////////////////////////
// Problem 4: TernaryTree Prefix Iterator

private:

    // push subtree of aNode [30]
    void push_subtrees(const TTree* aNode)
    {
        if (!aNode->getRight().empty())
        {
            fStack.push(&aNode->getRight());
        }
        if (!aNode->getMiddle().empty())
        {
            fStack.push(&aNode->getMiddle());
        }
        if (!aNode->getLeft().empty())
        {
            fStack.push(&aNode->getLeft());
        }
    }

public:

    // iterator constructor [12]
    TernaryTreePrefixIterator( const TTree* aTTree ) :
        fTTree(aTTree)
    {
        if (fTTree != &TTree::NIL)
        {
            fStack.push(fTTree);
        }
    }

    // iterator dereference [8]
    const T& operator*() const
    {
        return **fStack.top();
    }

    // prefix increment [12]
    Iterator& operator++()
    {
        if (!fStack.empty())
        {
            const TTree* current = fStack.top();
            fStack.pop();
            push_subtrees(current);
        }

        return *this;
    }

    // iterator equivalence [12]
    bool operator==(const Iterator& aOtherIter) const
    {
        return (fTTree == aOtherIter.fTTree)
            && (fStack == aOtherIter.fStack);
    }

    // auxiliaries [4,10]
    Iterator begin() const
    {
        return Iterator(fTTree);
    }

    Iterator end() const
    {
        Iterator lIter(fTTree);
        lIter.fStack = TTreeStack();
        return lIter;
    }
};

```