Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: 4, Binary Search Trees & In-Order Traversal
Due date: May 26, 2022, 14:30
Lecturer: Dr. Markus Lumpe

Your name: Nguyen Minh Duy **Your student id:** 104974743

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Tues 16:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30

Marker's comments:

Problem	Marks	Obtained
1	94	
2	42	
3	8+86=94	
Total	230	

Extension certification:

This assignment has been given an extension and is now due on _____

Signature of Convener:_____

Problem 1: BinaryTreeNode.h


```

// COS30008, Problem Set 4, Problem 1, 2022

#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;                                      // new parent

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

        if ( x->empty() )
        {
            return false;                               // delete failed
        }

        if ( !x->left->empty() )
        {
            const T& lKey = x->left->findMax();         // find max to left
            x->key = lKey;
            x->left->remove( lKey, x );
        }
        else
        {
            if ( !x->right->empty() )
            {
                const T& lKey = x->right->findMin();    // find min to right
                x->key = lKey;
                x->right->remove( lKey, x );
            }
            else
            {
                if ( y != &NIL )                        // y can be NIL
                {
                    if ( y->left == x )
                    {
                        y->left = &NIL;
                    }
                    else
                    {
                        y->right = &NIL;
                    }
                }

                delete x;                               // free deleted node
            }
        }

        return true;
```

```
    }

    // PS4 starts here

    BinaryTreeNode() : // Default constructor
        key(T()),
        left(&NIL),
        right(&NIL)
    {}

    BinaryTreeNode(const T& aKey) :  // Copy constructor
        key(aKey),
        left(&NIL),
        right(&NIL)
    {}

    BinaryTreeNode(T&& aKey) :// Move constructor
        key(std::move(aKey)),
        left(&NIL),
        right(&NIL)
    {}

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

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

    bool empty() const  // Check if node is empty
    {
        return this == &NIL;
    }

    bool leaf() const // Check if the elements is leaf or not
    {
        return left->empty() && right->empty();
    }

    size_t height() const
    {
        if (empty())
        {
```

```
            throw std::domain_error("Tree is empty. No height can be calculated.");
        }

        if (leaf())
        {
            return 0;
        }
        else
        {
            size_t leftHeight = 0;
            size_t rightHeight = 0;

            if (!left->empty())
            {
                leftHeight = left->height();    // đổi gốc
            }

            if (!right->empty())
            {
                rightHeight = right->height();   // đổi gốc
            }

            return max(leftHeight, rightHeight) + 1;
        }
    }

    bool insert(const T& aKey)
    {
        BTreeNode x = this;
        BTreeNode y = &NIL;

        while (!x->empty())
        {
            y = x;

            if (aKey == x->key)
            {
                return false;         // duplicate key - error
            }

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

        BTreeNode z = new BNode(aKey);

        if (y->empty())
        {
            return false;           // insert failed (NIL) - emptyy
```

```
        }
        else
        {
            if (aKey < y->key)
            {
                y->left = z;
            }
            else
            {
                y->right = z;
            }
        }

        return true;                // insert done
    }
};

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

```

Output

```
Microsoft Visual Studio Debu

Test BinaryTreeNode:
lRoot is NIL; insert failed successfully.
Determining height of NIL.
Successfuly caught domain error: Tree is empty. No height can be calculated.
Insert of 25 as root.
Successfully applied move constructor.
Insert of 10 succeeded.
Insert of 15 succeeded.
Insert of 37 succeeded.
Insert of 10 failed (duplicate key).
Insert of 30 succeeded.
Insert of 65 succeeded.
Height of tree: 2
Delete binary tree
Test BinaryTreeNode completed.

D:\0Study\0C30008 Data Structures_And_Patterns\Problem Set 4\Student_Project1\x64\Debug\
Student_Project1.exe (process 51976) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debuggin
g->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Problem 2: BinarySearchTree.h


```

// COS30008, Problem Set 4, Problem 2, 2022

#pragma once

#include "BinaryTreeNode.h"

#include <stdexcept>

// Problem 3 requirement
template<typename T>
class BinarySearchTreeIterator;

template<typename T>
class BinarySearchTree
{
private:

    using BNode = BinaryTreeNode<T>;
    using BTreeNode = BNode*;

    BTreeNode fRoot;

public:

    BinarySearchTree()
    {
        fRoot = &BNode::NIL;
    }

    ~BinarySearchTree()
    {
        if ( !empty() )
        {
            delete fRoot;
        }
    }

    bool empty() const
    {
        return fRoot == &BNode::NIL;
    }

    size_t height() const
    {
        // The empty tree has no height.
        if ( empty() )
        {
            throw std::domain_error( "Empty tree has no height." );
```

```
        }

        return fRoot->height();
    }

    bool insert( const T& aKey )
    {
        if ( empty() )
        {
            fRoot = new BNode( aKey );

            return true;
        }
        else
        {
            return fRoot->insert( aKey );
        }
    }

    bool remove( const T& aKey )
    {
        if ( empty() )
        {
            throw std::domain_error( "remove(): NIL encountered." );
        }

        if ( fRoot->leaf() )
        {
            // last node
            if ( fRoot->key == aKey )
            {
                delete fRoot;
                fRoot = &BNode::NIL;

                return true;
            }

            return false;
        }
        else
        {
            return fRoot->remove( aKey, &BNode::NIL );
        }
    }

    // Problem 3 methods

    using Iterator = BinarySearchTreeIterator<T>;

```

```
    // Allow iterator to access private member variables
    friend class BinarySearchTreeIterator<T>;

    Iterator begin() const
    {
        return Iterator( *this );
    }

    Iterator end() const
    {
        return begin().end();
    }
};

```

Output

```
Test Binary Search Tree:
Error: Empty tree has no height.
insert of 25 succeeded.
insert of 10 succeeded.
insert of 15 succeeded.
insert of 37 succeeded.
insert of 10 failed.
insert of 30 succeeded.
insert of 65 succeeded.
Height of tree: 2
Delete binary search tree now.
remove of 25 succeeded.
remove of 10 succeeded.
remove of 15 succeeded.
remove of 37 succeeded.
remove of 10 failed.
remove of 30 succeeded.
remove of 65 succeeded.
Test Binary Search Tree completed.

D:\0Study\0C30008 Data Structures_And_Patterns\Problem Set 4\
Student_Project1\x64\Debug\Student_Project1.exe (process 1178
0) exited with code 0.
To automatically close the console when debugging stops, enab
le Tools->Options->Debugging->Automatically close the console
 when debugging stops.
Press any key to close this window . . .
```

Problem 3: BinarySearchTreeIterator.h


```

// COS30008, Problem Set 4, Problem 3, 2022

#pragma once

#include "BinarySearchTree.h"

#include <stack>

template<typename T>
class BinarySearchTreeIterator
{
private:

    using BSTree = BinarySearchTree<T>;
    using BNode = BinaryTreeNode<T>;
    using BTreeNode = BNode*;
    using BTNStack = std::stack<BTreeNode>;

    const BSTree& fBSTree;
    BTNStack fStack;

    void pushLeft( BTreeNode aNode )
    {
        while ( !aNode->empty() )
        {
            fStack.push( aNode );
            aNode = aNode->left;
        }
    }

public:

    using Iterator = BinarySearchTreeIterator<T>;

    BinarySearchTreeIterator( const BSTree& aBSTree ) :
        fBSTree(aBSTree)
    {
        pushLeft( fBSTree.fRoot );
    }

    const T& operator*() const
    {
        return fStack.top()->key;
    }

    Iterator& operator++()
    {
        BTreeNode newNode = fStack.top()->right;
```

```
        fStack.pop();

        pushLeft(newNode);

        return *this;
    }

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

        ++(*this);

        return temp;
    }

    bool operator==( const Iterator& aOtherIter ) const
    {
        return
            &fBSTree == &aOtherIter.fBSTree &&
            fStack.size() == aOtherIter.fStack.size();
    }

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

    Iterator begin() const
    {
        Iterator iter = *this;

        iter.fStack = BTNStack();
        iter.pushLeft( iter.fRoot );

        return iter;
    }

    Iterator end() const
    {
        Iterator iter = *this;

        iter.fStack = BTNStack();

        return iter;
    }
};

```

Output