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: Tran Quoc Dung **Your student id:** 103803891

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 Set 4

File: BinaryTreeNode.h

```
#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() :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; // Cannot insert into NIL

        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;
```

File: BinarySearchTree.h

```
#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 (!fRoot->empty()) delete fRoot; }

   bool empty() const { return fRoot->empty(); }
   size_t height() const
   {
      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;
      }
      return fRoot->insert(aKey);

   }
   bool remove(const T& aKey)
   {
      if (empty()) throw std::domain_error("Cannot remove in empty tree.");
      if (fRoot->leaf())
      {
         if (fRoot->key != aKey) return false;
         fRoot = &BNode::NIL; return true;
      }
      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).begin(); }
    Iterator end() const { return Iterator(*this).end(); }
};
```

File: BinarySearchTreeIterator.h

```
#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;            // binary search tree
   BTNStack fStack;                  // DFS traversal stack

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

public:

   using Iterator = BinarySearchTreeIterator<T>;

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

   const T& operator*() const
   {
      return fStack.top()->key;
   }
   Iterator& operator++()
   {
      BTreeNode lPopped = fStack.top();
      fStack.pop();
      pushLeft(lPopped->right);
      return *this;
   }
   Iterator operator++(int)
   {
      Iterator lTmp = *this;
      ++(*this);
      return lTmp;
   }
```

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

    Iterator begin() const
    {
        Iterator lTmp = *this;
        lTmp.fStack = BTNStack();
        lTmp.pushLeft(lTmp.fBSTree.fRoot);
        return lTmp;
    }
    Iterator end() const
    {
        Iterator lTmp = *this;
        lTmp.fStack = BTNStack();
        return lTmp;
    }
};
```

File: Main.cpp

```
#include <iostream>

using namespace std;

#define P1
#define P2
#define P3

#ifdef P1

#include "BinaryTreeNode.h"

// operator<: order strings in binary tree
bool operator<(const string& aLHS, const string& aRHS)
{
   return aLHS.compare(aRHS) < 0;
}

void testBNode()
{
   using BTNode = BinaryTreeNode<string>;
   using BTTree = BTNode*;

   BTTree lRoot = &BTNode::NIL;

   cout << "Test BinaryTreeNode:" << endl;

   if (lRoot->insert("25"))
   {
      cerr << "This message must not appear! NIL cannot be used to insert elements." << endl;
   }
   else
   {
      cout << "lRoot is NIL; insert failed successfully." << endl;
   }

   try
   {
      cout << "Determining height of NIL." << endl;

      lRoot->height();

      cerr << "This message must not appear! NIL has no height." << endl;
   }
   catch (domain_error e)
   {
      cout << "Successfuly caught domain error: " << e.what() << endl;
   }

   string lValues[] = { "10", "15", "37", "10", "30", "65" };
   string l25("25");

   cout << "Insert of " << l25 << " as root." << endl;

   lRoot = new BTNode(std::move(l25));
```

```
    if (l25.empty())
    {
        cout << "Successfully applied move constructor." << endl;
    }
    else
    {
        cerr << "This message must not appear! Move failed." << endl;
    }

    for (const string& i : lValues)
    {
        if (lRoot->insert(i))
        {
            cout << "Insert of " << i << " succeeded." << endl;
        }
        else
        {
            cout << "Insert of " << i << " failed (duplicate key)." << endl;
        }
    }

    try
    {
        cout << "Height of tree: " << lRoot->height() << endl;
    }
    catch (domain_error e)
    {
        cerr << "This message must not appear! lRoot is not NIL." << endl;
        cerr << e.what() << endl;
    }

    cout << "Delete binary tree" << endl;

    if (!lRoot->empty())
    {
        delete lRoot;
    }
    else
    {
        cerr << "This message must not appear!" << endl;
    }

    cout << "Test BinaryTreeNode completed." << endl;
}

#endif

#ifdef P2

#include "BinarySearchTree.h"

void testBinarySearchTree()
{
    using BSTree = BinarySearchTree<int>;

    cout << "Test Binary Search Tree:" << endl;
```

```
    BSTree lTree;
    int lValues[] = { 25, 10, 15, 37, 10, 30, 65 };

    try
    {
        lTree.height();

        cout << "Height on empty tree succeeded!" << endl;
    }
    catch (domain_error e)
    {
        cerr << "Error: " << e.what() << endl;
    }

    for (const int& i : lValues)
    {
        if (lTree.insert(i))
        {
            cout << "insert of " << i << " succeeded." << endl;
        }
        else
        {
            cout << "insert of " << i << " failed." << endl;
        }
    }

    cout << "Height of tree: " << lTree.height() << endl;

    cout << "Delete binary search tree now." << endl;

    for (const int& i : lValues)
    {
        if (lTree.remove(i))
        {
            cout << "remove of " << i << " succeeded." << endl;
        }
        else
        {
            cout << "remove of " << i << " failed." << endl;
        }
    }

    cout << "Test Binary Search Tree completed." << endl;
}

#endif

#ifdef P3

#include "BinarySearchTreeIterator.h"

void testIterator()
{
    using BSTree = BinarySearchTree<int>;

    cout << "Test Binary Search Tree Iterator DFS:" << endl;
```

```
    BSTree lTree;
    int lValues[] = { 25, 10, 15, 37, 10, 30, 65, 8 };

    for (const int& i : lValues)
    {
        lTree.insert(i);
    }

    cout << "DFS:";

    for (const auto& i : lTree)
    {
        cout << " " << i;
    }

    cout << endl;

    cout << "Test Binary Search Tree Iterator DFS completed." << endl;
}

#endif

int main()
{
#ifdef P1

    cout << "Problem 1:" << endl;
    testBNode();
    cout << "\n" << endl;

#endif

#ifdef P2

    cout << "Problem 2:" << endl;
    testBinarySearchTree();
    cout << "\n" << endl;

#endif

#ifdef P3

    cout << "Problem 3:" << endl;
    testIterator();
    cout << "\n" << endl;

#endif

    return 0;
}
```