Swinburne University Of Technology

Faculty of Information and Communication Technologies

ASSIGNMENT COVER SHEET

Subject Code:	HIT3303/8303
Subject Title:	Data Structures & Patterns
Assignment number and title:	7 – Tree Traversal
Due date:	**May 25, 2011, 10:30 a.m., on paper**
Lecturer:	Dr. Markus Lumpe

Your name:____________________

Marker's comments:

Problem	Marks	Obtained
1	115	
Total	115	

Extension certification:

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

Signature of Convener:________________________

NTree.h

```cpp
#ifndef NTREE_H_
#define NTREE_H_

#include <stdexcept>

#include <iostream>

#include "TreeVisitor.h"
#include "DynamicQueue.h"

template<class T, int N>
class NTree
{
private:
	const T* fKey;
	NTree<T,N>* fNodes[N];

	NTree(): fKey((T*)0)
	{
		for(int i = 0; i < N; i++)
			fNodes[i] = (NTree<T,N>*)0;
	}

public:
	static NTree<T,N> NIL;

	NTree(const T& aKey): fKey(&aKey)
	{
		for(int i = 0; i < N; i++)
			fNodes[i] = &NIL;
	}

	NTree(const NTree<T,N>& aOtherNTree)
	{
		for(int i = 0; i < N; i++)
			fNodes[i] = &NIL;
		fKey = aOtherNTree.fKey;

		for(int i = 0; i < N; i++){
			if(!aOtherNTree[i].isEmpty()){
				NTree<T,N>* temp = new NTree(aOtherNTree[i]);
				attachNTree(i, temp);
			}
		}
	}

	NTree& operator=(const NTree<T,N>& aOtherNTree)
	{
		//delete old elements
		for(int i = 0; i < N; i++){
			if(fNodes[i] != &NIL){
				delete fNodes[i];
			}
		}
		//create new ones
		for(int i = 0; i < N; i++)
			fNodes[i] = &NIL;
		fKey = aOtherNTree.fKey;

		for(int i = 0; i < N; i++){
			if(!aOtherNTree[i].isEmpty()){
				NTree<T,N>* temp = new NTree(aOtherNTree[i]);
				attachNTree(i, temp);
				}
			}
		}

		~NTree()
		{
			for(int i = 0; i < N; i++){
				if(fNodes[i] != &NIL){
					delete fNodes[i];
				}
			}
		}

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

		const T& key() const
		{
			if(isEmpty())
				throw std::domain_error("Empty tree!");

			return *fKey;
		}

		NTree& operator[](unsigned int aIndex) const
		{
			if(isEmpty())
				throw std::domain_error("Empty NTree!");

			if((aIndex >= 0) && (aIndex < N)){
				return *fNodes[aIndex];
			}
			else
				throw std::out_of_range("Illegal index!");
		}

		void attachNTree(unsigned int aIndex, NTree<T,N>* aNTree)
		{
			if(isEmpty())
				throw std::domain_error("Empty tree!");

			if((aIndex >= 0) && (aIndex < N)){
				if(fNodes[aIndex] != &NIL)
					throw std::domain_error("Non-empty subtree!");

				fNodes[aIndex] = aNTree;
			}
			else
				throw std::out_of_range("Index out of range!");
		}

		NTree* detachNTree(unsigned int aIndex)
		{
			if(isEmpty())
				throw std::domain_error("Empty tree!");

			if((aIndex >= 0) && (aIndex < N)){
				NTree<T,N>* returnTree = fNodes[aIndex];
				fNodes[aIndex] = &NIL;
				return returnTree;
			}
			else
				throw std::out_of_range("Illegal index!");
		}
		void doDepthFirstTraversal(const TreeVisitor<T>& aVisitor) const
		{
			if(!isEmpty()){
				aVisitor.preVisit(key());
				for(int i = 0; i < N; i++){
					if(fNodes[i] != &NIL){
						fNodes[i]->doDepthFirstTraversal(aVisitor);
						aVisitor.inVisit(key());
					}
				}
				aVisitor.postVisit(key());
			}
		}

		void doBreadthFirstTraversal(const TreeVisitor<T>& aVisitor) const
		{
			DynamicQueue<NTree<T,N> > lQueue;

			if(!isEmpty())
				lQueue.enqueue(*this);

			while(!lQueue.isEmpty()){
				const NTree<T,N>& head = lQueue.dequeue();
				if(!head.isEmpty())
					aVisitor.visit(head.key());
				for(int i = 0; i < N; i++){
					if(!head[i].isEmpty())
						lQueue.enqueue(head[i]);
				}
			}
		}
};

template<class T, int N>
NTree<T,N> NTree<T,N>::NIL;

#endif /* NTREE_H_ */
```