

**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: \_\_\_\_\_

## Problem Set 6: Tree Traversal

### Problem 1:

Using the template class `DynamicQueue` defined in problem set 6 and the tree traversal classes, implement the template class `NTree` as specified below:

```
#include <stdexcept>

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

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

    NTree();

public:
    static NTree<T,N> NIL;

    NTree( const T& aKey );
    NTree( const NTree<T,N>& aOtherNTree );
    NTree& operator=( const NTree<T,N>& aOtherNTree );
    ~NTree();

    bool isEmpty() const;
    const T& key() const;

    NTree& operator[]( unsigned int aIndex ) const;

    void attachNTree( unsigned int aIndex, NTree<T,N>* aNTree );
    NTree* detachNTree( unsigned int aIndex );

    void doDepthFirstTraversal( const TreeVisitor<T>& aVisitor ) const;
    void doBreadthFirstTraversal( const TreeVisitor<T>& aVisitor ) const;
};
```

Use the supplied file `"TreeVisitor.h"` and implement the required behavior. Use the solution of problem 6 and verify your results with the supplied test harnesses.

Please note that in contrast to the binary tree traversal, we cannot support in-order traversal for N-ary trees (i.e., trees with  $N > 2$ ). Therefore, the solution will only require a proper handling of pre-order and post-order tree traversal. However, the approach for tree traversal of N-ary trees follows the scheme for binary trees shown in class.

## Test harness 1:

```

void test1()
{
    string A( "A" );
    string A1( "AA" );
    string A2( "AB" );
    string A3( "AC" );
    string AA1( "AAA" );
    string AB1( "ABA" );
    string AB2( "ABB" );

    typedef NTree<string,3> NS3Tree;

    NS3Tree root( A );

    root.attachNTree( 0, new NS3Tree( A1 ) );
    root.attachNTree( 1, new NS3Tree( A2 ) );
    root.attachNTree( 2, new NS3Tree( A3 ) );

    root[0].attachNTree( 0, new NS3Tree( AA1 ) );
    root[1].attachNTree( 0, new NS3Tree( AB1 ) );
    root[1].attachNTree( 1, new NS3Tree( AB2 ) );

    cout << "root:      " << root.key() << endl;
    cout << "root[0]:    " << root[0].key() << endl;
    cout << "root[1]:    " << root[1].key() << endl;
    cout << "root[2]:    " << root[2].key() << endl;
    cout << "root[0][0]: " << root[0][0].key() << endl;
    cout << "root[1][0]: " << root[1][0].key() << endl;
    cout << "root[1][1]: " << root[1][1].key() << endl;

    NS3Tree copy = root;

    cout << "copy:      " << copy.key() << endl;
    cout << "copy[0]:   " << copy[0].key() << endl;
    cout << "copy[1]:   " << copy[1].key() << endl;
    cout << "copy[2]:   " << copy[2].key() << endl;
    cout << "copy[0][0]: " << copy[0][0].key() << endl;
    cout << "copy[1][0]: " << copy[1][0].key() << endl;
    cout << "copy[1][1]: " << copy[1][1].key() << endl;

    NS3Tree* temp = copy.detachNTree( 0 );

    if ( &copy[0] == &NS3Tree::NIL )
        cout << "Detach succeeded." << endl;
    else
        cout << "Detach failed." << endl;

    delete temp;
}

```

**Result:**

```
root:      A
root[0]:   AA
root[1]:   AB
root[2]:   AC
root[0][0]: AAA
root[1][0]: ABA
root[1][1]: ABB
copy:      A
copy[0]:   AA
copy[1]:   AB
copy[2]:   AC
copy[0][0]: AAA
copy[1][0]: ABA
copy[1][1]: ABB
Detach succeeded.
```

**Test harness 2:**

```

void test2()
{
    string A( "A" );
    string A1( "AA" );
    string A2( "AB" );
    string A3( "AC" );
    string AA1( "AAA" );
    string AB1( "ABA" );
    string AB2( "ABB" );

    typedef NTree<string,3> NS3Tree;

    NS3Tree root( A );

    root.attachNTree( 0, new NS3Tree( A1 ) );
    root.attachNTree( 1, new NS3Tree( A2 ) );
    root.attachNTree( 2, new NS3Tree( A3 ) );

    root[0].attachNTree( 0, new NS3Tree( AA1 ) );
    root[1].attachNTree( 0, new NS3Tree( AB1 ) );
    root[1].attachNTree( 1, new NS3Tree( AB2 ) );

    PreOrderVisitor<string> v1;
    PostOrderVisitor<string> v2;
    TreeVisitor<string> v3;

    cout << "Pre-order traversal:" << endl;
    root.doDepthFirstTraversal( v1 );
    cout << endl;

    cout << "Post-order traversal:" << endl;
    root.doDepthFirstTraversal( v2 );
    cout << endl;

    cout << "Breadth-first traversal:" << endl;
    root.doBreadthFirstTraversal( v3 );
    cout << endl;
}

```

**Result:**

```

Pre-order traversal:
A AA AAA AB ABA ABB AC
Post-order traversal:
AAA AA ABA ABB AB AC A
Breadth-first traversal:
A AA AB AC AAA ABA ABB

```

**Submission deadline: Wednesday, May 25, 2011, 10:30 a.m.**

**Submission procedure: on paper.**