Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures & Patterns
Assignment number and title: 7 – NTree Copy Control & Breadth-first Traversal
Due date: May 23, 2017, 14:30
Lecturer: Dr. Markus Lumpe

Your name:_____ **Your student id:**_____

Check Tutorial	Wed 10:30	Wed 12:30	Thu 13:30	Thu 15:30	Fri 13:30	Fri 15:30

Marker's comments:

Problem	Marks	Obtained
1 – copy control	12	
2 – tree traversal	16	
Total	28	

Extension certification:

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

Signature of Convener:_____

Problem Set 7: NTree Copy Control & Breadth-first Traversal

Using the template class `DynamicQueue` defined in problem set 6 and the template class `NTree` defined in tutorial 11, add copy control and breadth-first traversal as specified below:

```
#pragma once

#include <stdexcept>

#include "TreeVisitor.h"

template<class T, int N>
class NTree
{
private:
  const T* fKey;              // 0 for empty NTree
  NTree<T,N>* fNodes[N];      // N subtrees of degree N

  NTree();                    // sentinel constructor

public:
  static NTree<T,N> NIL;      // sentinel

  NTree( const T& aKey );     // a simple NTree with key and N subtrees

  bool isEmpty() const;       // is tree empty
  const T& key() const;       // get key (node value)

  // indexer (allow for result modification by client - no const in result)
  NTree& operator[]( unsigned int aIndex ) const;

  // tree manipulators (using constant references)
  void attachNTree( unsigned int aIndex, const NTree<T,N>& aNTree );
  const NTree& detachNTree( unsigned int aIndex );

  // depth-first traversal
  void traverseDepthFirst( const TreeVisitor<T>& aVisitor ) const;

  // copy control
  NTree( const NTree& aOtherNTree );
  ~NTree();
  NTree& operator=( const NTree& aOtherNTree );

  // breadth-first traversal
  void traverseBreadthFirst( const TreeVisitor<T>& aVisitor ) const;
};
```

Use "`TreeVisitor.h`" available on Blackboard and implement the breadth-first traversal.

You need a local queue variable in `traverseBreadthFirst`. To avoid unwanted copying, use a pointer to **`const`** `NTree<T,N>` as type for the required `DynamicQueue` value object. That is, specify `DynamicQueue<`**`const`**` NTree<T,N>*> lQueue`, if `lQueue` is the local queue object in `traverseBreadthFirst`.

Test harness:

```
void testNTreeCopyControl()
{
  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 )) );

  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;

  // test traversal
  TreeVisitor<string> v4;
  cout << "Breadth-first traversal:" << endl;
  root.traverseBreadthFirst( v4 );
  cout << endl;

  cout << "Success." << endl;

  return 0;
}
```

Result:

```
copy:       A
copy[0]:    AA
copy[1]:    AB
copy[2]:    AC
copy[0][0]: AAA
copy[1][0]: ABA
copy[1][1]: ABB
Breadth-first traversal:
A AA AB AC AAA ABA ABB
Success.
```

Submission deadline: Tuesday, May 23, 2017, 14:30.

Submission procedure: on paper on paper (printout of NTree copy control and breath-first traversal).