Swinburne University of Technology

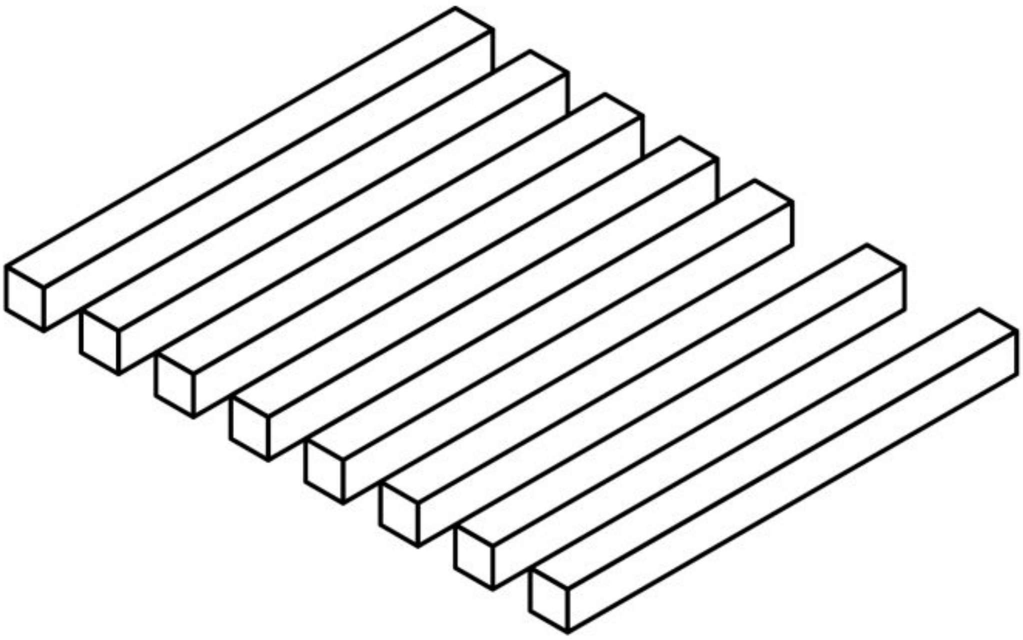
Faculty of Science, Engineering and Technology

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	11, BTrees
Lecturer:	Dr. Markus Lumpe

If you think it's simple, then you have misunderstood the problem.

Bjarne Stroustrup



BTrees

We study the construction of general n-ary trees in the class this week. The aim of this tutorial is to define a template class `BTree` that implements the basic infrastructure of binary trees, including full copy control.

The lecture material discusses most of the implementation details. However, some of the features (i.e., methods) defined for template class `NTree` have to be adjusted in order to obtain a suitable binary tree implementation. Just creating a type alias for `NTree`, in which `N=2`, does not suffice as it would not provide us with an abstraction that is conceptually close enough to the hierarchical data structure binary tree.

```
#pragma once

#include <stdexcept>

template<typename T>
class BTree
{
private:
  T fKey;                                        // T() for empty BTree
  BTree<T>* fLeft;
  BTree<T>* fRight;

  BTree();                                       // sentinel constructor

  // tree manipulator auxiliaries
  void attach( BTree<T>** aNode, const BTree<T>& aBTree );
  const BTree<T>& detach( BTree<T>** aNode );

public:
  static BTree<T> NIL;                           // Empty BTree

  BTree( const T& aKey );                        // BTree leaf
  BTree( T&& aKey );                             // BTree leaf

  BTree( const BTree& aOtherBTree );             // copy constructor
  BTree( BTree&& aOtherBTree );                  // move constructor

  virtual ~BTree();                              // destructor

  BTree& operator=( const BTree& aOtherBTree ); // copy assignment operator
  BTree& operator=( BTree&& aOtherBTree );      // move assignment operator

  virtual BTree* clone();                        // clone a tree

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

  const BTree& left() const;
  const BTree& right() const;

  // tree manipulators
  void attachLeft( const BTree<T>& aBTree );
  void attachRight( const BTree<T>& aBTree );
  const BTree& detachLeft();
  const BTree& detachRight();
};
```

Remember the canonical elements of copy control and how those elements are connected. The destructor is responsible for releasing resources. Here, the resources are the tree nodes. Empty binary subtrees must not be destroyed. We do not "own" empty binary trees.

The copy constructor has to perform a deep-copy of the argument and initialize this tree object. The assignment operator is a combination of destructor and copy constructor. In contrast to a copy constructor, the assignment operator changes an initialized object. We have to release resources first. Also, make sure that the assignment operator is protected against "accidental suicide" of the tree object.

The methods that copy tree objects must guarantee that `NIL` is not copied. `NIL` is a singleton and creating additional instances of `NIL` would break the sentinel protocol.

Implement the template class `BTree` in three stages:

1. Define the basic infrastructure.
    - All methods except copy and move operations
    - Provide a clone method with an empty body (to allow for compilation).

   You can use `#define P1` in `Main.cpp` to enable the corresponding test driver.

   Result:

```
Test basic semantics.
root:       Hello World!
root->L:    A
root->R:    B
root->L->L: AA
root->R->R: BB
All trees are going to be deleted now!
```

2. Define copy control:
    - Copy constructor
    - Copy assignment
    - Clone

   You can use `#define P2` in `Main.cpp` to enable the corresponding test driver.

   Result:

```
Test copy semantics.
root:       Hello World!
root->L:    A
root->R:    B
root->L->L: AA
root->R->R: BB
Illegal binary tree operation.
copy:       Hello World!
copy->L:    A
copy->R:    B
copy->L->L: AA
copy->R->R: BB
Illegal binary tree operation.
root1:       Hello World!
root1->L:    A
root1->R:    B
root1->L->L: AA
root1->R->R: BB
clone:       Hello World!
clone->L:    A
clone->R:    B
clone->L->L: AA
clone->R->R: BB
All trees are going to be deleted now!
```

3. Define move semantics:
    - Move constructor
    - Move assignment

   You can use `#define P3` in `Main.cpp` to enable the corresponding test driver.

   Result:

```
Test move semantics.
root:       Hello World!
root->L:    A
root->R:    B
root->L->L: AA
root->R->R: BB
Illegal binary tree operation.
root:
copy:       Hello World!
copy->L:    A
copy->R:    B
copy->L->L: AA
copy->R->R: BB
Illegal binary tree operation.
copy:
root1:       Hello World!
root1->L:    A
root1->R:    B
root1->L->L: AA
root1->R->R: BB
All trees are going to be deleted now!
```

Please check with the tutor. You should complete this task as it may be a prerequisite for a later one.