Swinburne University of Technology

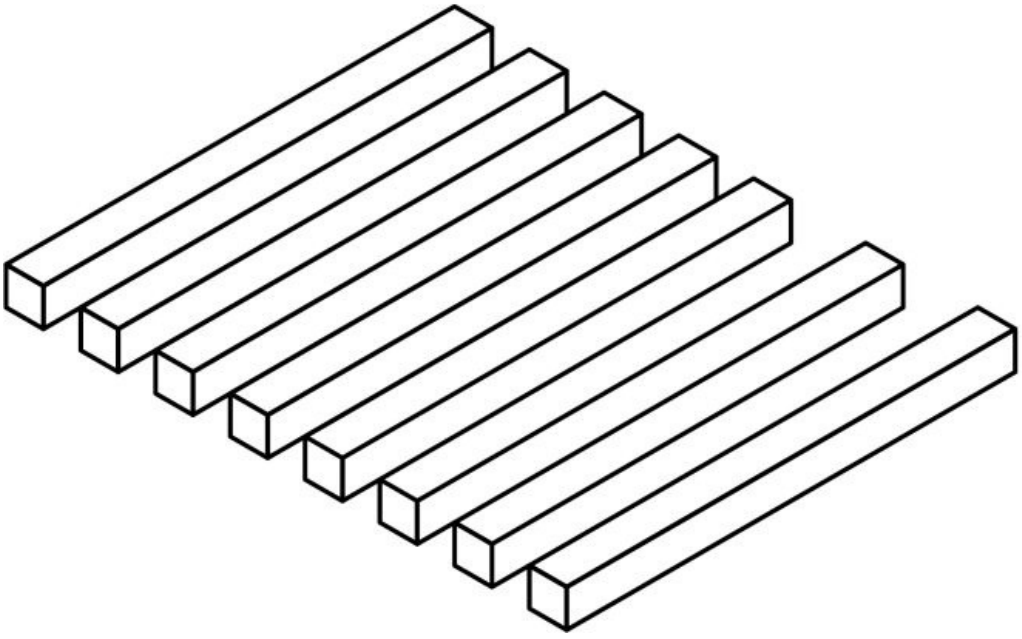
School of Science, Computing and Engineering Technologies

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	10, Binary Trees
Lecturer:	Ms. Siti Hawa

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

Bjarne Stroustrup



Binary Trees

We have studied the construction of general n-ary trees in the class last 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 and recursive tree traversal.

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 <memory>
#include <cassert>
#include <iostream>

#include "Visitors.h"

template<typename T>
class BTree
{
public:
  using Node = std::unique_ptr<BTree>;

  BTree(const T& aKey = T{}) noexcept;
  BTree(T&& aKey) noexcept;

  ~BTree()
  {
    std::cout << "Delete " << fKey << std::endl;
  }

  template<typename... Args>
  static Node makeNode(Args&&... args);

  // copy semantics
  BTree(const BTree& aOther);
  BTree& operator=(const BTree& aOther);

  // move semantics
  BTree(BTree&& aOther) noexcept;
  BTree& operator=(BTree&& aOther) noexcept;

  void swap(BTree& aOther) noexcept;

  const T& operator*() const noexcept;
  bool hasLeft() const noexcept;
  BTree& left() const;
  bool hasRight() const noexcept;
  BTree& right() const;

  void attachLeft(Node& aNode);
  void attachRight(Node& aNode);

  Node detachLeft();
  Node detachRight();
```

```
  bool leaf() const noexcept;
  size_t height() const noexcept;

  const T& findMax() const noexcept;
  const T& findMin() const noexcept;

  void doDepthFirstSearch(const TreeVisitor<T>& aVisitor) const noexcept;

private:
  T fKey;
  Node fLeft;
  Node fRight;
};
```

Implement the template class `BTree`.

The test driver in `Main.cpp` should produce the following output:

```
Pre-Order Traversal: 25 10 15 37 30 65
In-Order Traversal: 10 15 25 30 37 65
Post-Order Traversal: 15 10 30 65 37 25
Delete 25
Delete 37
Delete 65
Delete 30
Delete 10
Delete 15
```