

**Swinburne University of Technology***School of Science, Computing and Engineering Technologies***LABORATORY COVER SHEET**

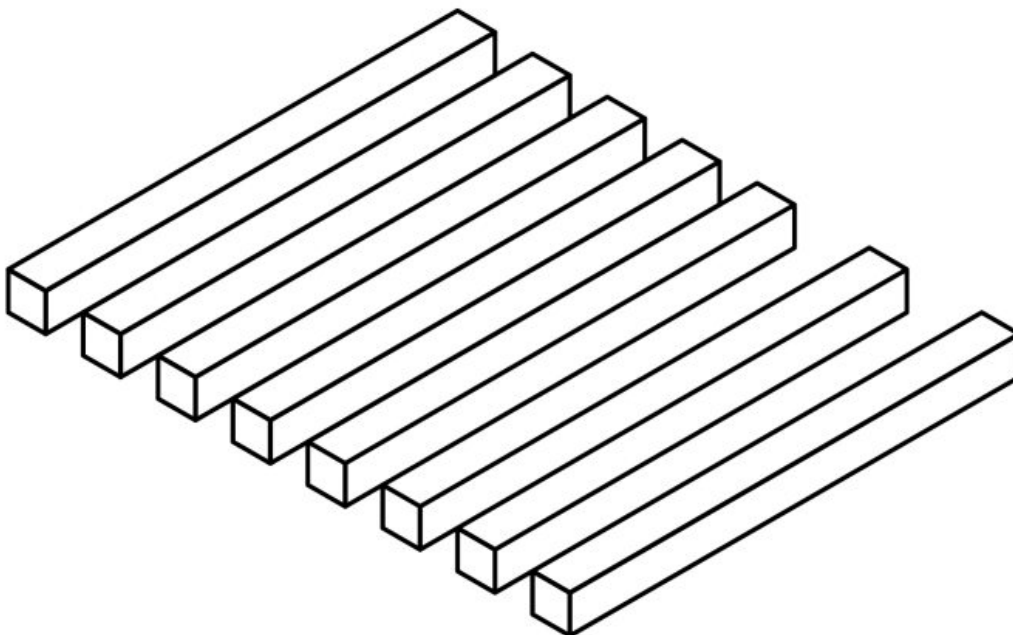
---

<b>Subject Code:</b>	COS30008
<b>Subject Title:</b>	Data Structures and Patterns
<b>Lab number and title:</b>	11, Binary Trees
<b>Lecturer:</b>	Dr. Markus Lumpe

---

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

**Bjarne Stroustrup**



## Binary Trees

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

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;

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

Implement the template class `BTree`.

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

```
Constructed nodes:
n25: 25
n10: 10
n15: 15
n37: 37
n30: 30
n65: 65
Node integrity (unique_ptr):
n10 ok
n37 ok
n15 ok
n30 ok
n65 ok
Service functions:
Height of aBTree: 2
Left max of 25: 15
Right min of 25: 30
Node hierarchy:
c25: 25
n10: 10
n15: 15
n37: 37
n30: 30
n65: 65
Detached nodes:
n25: 25
n10: 10
n15: 15
n37: 37
n30: 30
n65: 65
BTree test complete.
Delete 25
Delete 37
Delete 65
Delete 30
Delete 10
Delete 15
Delete 0
Delete 65
Delete 30
Delete 37
Delete 15
Delete 10
Delete 25
```

Can you explain the result?