Swinburne University Of Technology

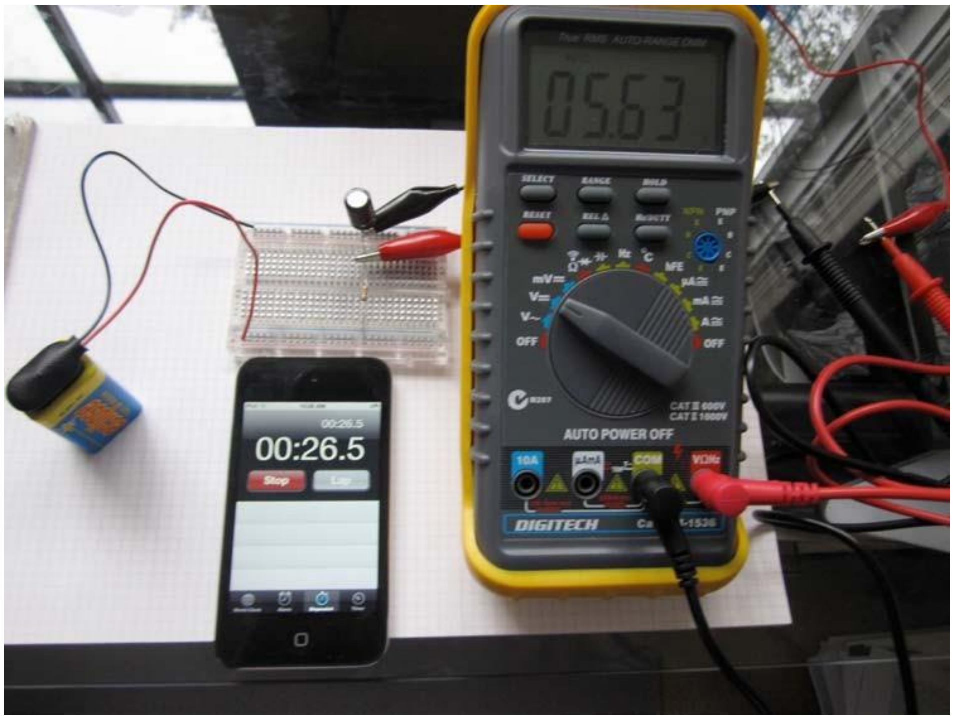
Faculty of Science, Engineering and Technology

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	9, Testing & Debugging
Lecturer:	Dr. Markus Lumpe

A journey of a thousand miles begins with a single step.

Lao Tsu



Testing & Debugging

Consider the code on the following pages. Somewhere there are bugs that will cause the program to crash. This is a serious problem and we need to fix it immediately. Apply debugging to locate the problem and devise a solution. This experiment requires code comprehension and debugging code, two elements of professional software development.

Answer the following questions:

1. Where is the problem?
2. What is a proper problem solution (i.e., bug fix)?
3. What causes the problem?

This is based on Lab 8 (BTree). Go through the codes of Lab 8, and try to debug this on paper without an ide first.

The faulty codes

BTree.h:

```
#pragma once
#include <stdexcept>
#include "TreeVisitor.h"

template<class T> class BTree
{
private:
     const T* fKey;
     BTree<T>* fLeft;
     BTree<T>* fRight;
     BTree() :fKey((T*)0)
     {
            fLeft = &NIL;
            fRight = &NIL;
     }

public:
     static BTree<T> NIL;

     BTree(const T& aKey) :fKey(&aKey)
     {
            fLeft = &NIL;
            fRight = &NIL;
     }

     ~BTree()
     {
            if (fLeft != &NIL)
                  delete fLeft;
            if (fRight != &NIL)
                  delete fRight;
     }

     bool isEmpty() const
     {
            return this == &NIL;
     }

     const T& key() const
```

```
    {
            if (isEmpty())
                    throw std::domain_error("Empty BTree");
            return *fKey;
    }

    BTree& left() const
    {
            if (isEmpty())
                    throw std::domain_error("Empty BTree");
            return *fLeft;
    }

    BTree& right() const
    {
            if (isEmpty())
                    throw std::domain_error("Empty BTree");
            return *fRight;
    }

    void attachRight(BTree<T>* aBTree)
    {
            if (isEmpty())
                    throw std::domain_error("Empty Tree");

            if (fRight != &NIL)
                    throw std::domain_error("Non-empty sub tree");

            fRight = aBTree;
    }

    void attachLeft(BTree<T>* aBTree)
    {
            if (isEmpty())
                    throw std::domain_error("Empty Tree");

            if (fLeft != &NIL)
                    throw std::domain_error("Non-empty sub tree");

            fLeft = aBTree;
    }

    BTree* detachLeft()
    {
            if (isEmpty())
                    throw std::domain_error("Empty Tree");
            BTree<T> Result = *fLeft;
            fLeft = &NIL;
            return &Result;
    }

    BTree* detachRight()
    {
            if (isEmpty())
                    throw std::domain_error("Empty Tree");
            BTree<T> Result = *fRight;
            fRight = &NIL;
            return &Result;
    }

    void transverseDepthFirst(const TreeVisitor<T>& aVisitor) const
    {
            if (!isEmpty())
            {
                  aVisitor.preVisit(key());
                  left().transverseDepthFirst(aVisitor);
                  aVisitor.inVisit(key());
                  right().transverseDepthFirst(aVisitor);
                  aVisitor.postVisit(key());
            }
      }
};

template<class T>
BTree<T> BTree<T>::NIL;
```

TreeVisitor.h:

```cpp
#pragma once
#include <iostream>

template<class T> class TreeVisitor
{
public:
     virtual ~TreeVisitor() {}
     virtual void preVisit(const T& aKey) const {}
     virtual void postVisit(const T& aKey) const {}
     virtual void inVisit(const T& aKey) const {}

     virtual void visit(const T& aKey) const
     {
            cout << aKey << " ";
     }
};

template<class T>
class PreOrderVisitor : public TreeVisitor<T>
{
     virtual void preVisit(const T& aKey) const
     {
            this->visit(aKey);
     }

};

template<class T>
class PostOrderVisitor : public TreeVisitor<T>
{
     virtual void PostVisit(const T& aKey) const
     {
            this->visit(aKey);
     }

};

template<class T>
class InOrderVisitor : public TreeVisitor<T>
{
     virtual void inVisit(const T& aKey) const
     {
            this->visit(aKey);
     }

};
```

main.cpp:

```
//test harness
#include <iostream>
#include "BTree.h"
#include <string>
#include "TreeVisitor.h"
using namespace std;

int main()
{
	string s1("Hello world!");
	string s2("A");
	string s3("B");
	string s4("C");
	BTree<string> A2Tree(s1);
	BTree<string> STree1(s2);
	BTree<string> STree2(s3);
	BTree<string> STree3(s4);

	A2Tree.attachLeft(&STree1);
	A2Tree.attachRight(&STree2);
	A2Tree.left().attachLeft(&STree3);

	cout << "Key:" << A2Tree.key() << endl;
	cout << "Key:" << A2Tree.left().left().key() << endl;

	A2Tree.transverseDepthFirst(PreOrderVisitor<string>());
	cout << endl;

	cout << A2Tree.detachLeft()->key() << endl;
	cout << A2Tree.left().detachLeft()->key() << endl;
	cout << A2Tree.detachRight()->key() << endl;

	return 0;
}
```