Tutorial/Lab 8

Conduct the experiment on binary tree traversal based on the NTree class described in the Lecture 8 (Slides 15 - 23) and TreeVisitor and BTree class described in Lecture 9 (Slides 6 – 11).

1. Add TreeVisitor.h below to your project, and completely implement the binary tree class BTree.h below.

TreeVisitor.h

```
#pragma once
#include <iostream>

template <class T>
class TreeVisitor
{
public:
	virtual ~TreeVisitor() {} //virtual default destructor
	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
	{
		std::cout << aKey << " ";
	}
};

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


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


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

};
```

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(); //empty BTree, (complete this)

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

	BTree(const T& aKey); //(complete this)
	~BTree(); //(complete this)

	bool isEmpty() const; //(complete this)
	const T& key() const; //(complete this)

	BTree& left() const; //(complete this)

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

	void attachLeft(BTree<T>* aBTree); //(complete this)

	void attachRight(BTree<T>* aBTree) {
		if (isEmpty())
			throw std::domain_error("Empty BTree");
		if (fRight != &NIL)
			throw std::domain_error("Non-empty sub tree");
		fRight = new BTree<T>(*aBTree);//makes allocation on heap
	}

	BTree* detachLeft(); //(complete this)

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

	void transverseDepthFirst(const TreeVisitor<T>& aVisitor) const;
	//(complete this)
};

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

2. Test the BTree and TreeVisitor classes with the test harness shown on the screenshot in Lecture 9 Slide 11.