Problem Set 4

Submission Instruction and Requirement:

Due date: 11:59pm 24th May 2019

1) Create respective folders for C++ source codes of each question and zip the folders of tasks you have attempted in one zip file. **Do not** include any Microsoft Visual Studio solution files in your submission.
2) Name the file in the pattern of studentid.yourname.ps4.zip OR. studentid.yourname.ps4.rar
3) Write a report on the codes you have created or on the task attempted and include the report in the zip/rar file mentioned in 2) with screenshots of successful running of your codes. If the code does not work as expected, please provide justifications.
4) The codes should be neat and be well-commented.
5) You code should be workable and without any error, warning message, infinite loop or any malicious function.
6) Submit the zip/rar file to Blackboard on time.

Task 1 (10 marks)

This task requires you to implement an N-ary Tree class. The TreeVisitor class and a test harness have been provided in this document to test your N-ary Tree. You may base your implementation on the solutions provided in Lab 8 and Lecture 8.

Complete the NTree.h provided below.
Include a depth first traversal method in your NTree class.

NTree.h

```
template<class T, int N>
class NTree
{
private:
	const T* fKey; // 0 for empty
	NTree <T, N>* fNodes[N]; // N subtrees of degree N
	NTree();

public:
	static NTree<T, N> NIL; // sentinel

	NTree(const T& aKey);
	~NTree();

	bool isEmpty() const;
	const T& key() const;

	NTree& operator[](int aIndex) const;
	void attachNTree(aIndex, NTree<T, N>* aNTree);
	NTree* detachNTree(int aIndex);
};
```

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);
	}
};
```

Please note that in contrast to the binary tree traversal, we cannot support in-order traversal for N-ary trees (i.e., trees with N > 2). Therefore, the solution will only require a proper handling of pre-order and post-order tree traversal.

However, the approach for tree traversal of N-ary trees follows the scheme for binary trees shown in class.

Test harness:

```
int main() {
	string A("A");
	string A1("AA");
	string A2("AB");
	string A3("AC");
	string AA1("AAA");
	string AB1("ABA");
	string AB2("ABB");
	typedef NTree<string, 3> NS3Tree;

	NS3Tree root(A);

	NS3Tree nodeA1(A1);
	NS3Tree nodeA2(A2);
	NS3Tree nodeA3(A3);
	NS3Tree nodeAA1(AA1);
	NS3Tree nodeAB1(AB1);
	NS3Tree nodeAB2(AB2);
	root.attachNTree(0, &nodeA1);
	root.attachNTree(1, &nodeA2);
	root.attachNTree(2, &nodeA3);
	root[0].attachNTree(0, &nodeAA1);
	root[1].attachNTree(0, &nodeAB1);
	root[1].attachNTree(1, &nodeAB2);
	cout << "root: " << root.key() << endl;
	cout << "root[0]: " << root[0].key() << endl;
	cout << "root[1]: " << root[1].key() << endl;
	cout << "root[2]: " << root[2].key() << endl;
	cout << "root[0][0]: " << root[0][0].key() << endl;
	cout << "root[1][0]: " << root[1][0].key() << endl;
	cout << "root[1][1]: " << root[1][1].key() << endl;

	// test traversal
	PreOrderVisitor<string> v1;
	PostOrderVisitor<string> v2;
	cout << "Pre-order traversal:" << endl;
	root.traverseDepthFirst(v1);
	cout << endl;
	cout << "Post-order traversal:" << endl;
	root.traverseDepthFirst(v2);
	cout << endl;

	// test detachNTree
	root[0].detachNTree(0);
	root[1].detachNTree(0);
	root[1].detachNTree(1);
	root.detachNTree(0);
	root.detachNTree(1);
	root.detachNTree(2);
	if (&root[0] == &NS3Tree::NIL)
		cout << "Detach succeeded." << endl;
	else
		cout << "Detach failed." << endl;
}
```

Task 2 (4 marks)

Draw the tree generated in Task 1. Label each node based on the string assigned to it. I.e. Root will be labeled as Node “A”.

Create the table below in your report. Fill out the answers for each node.

Node	**Depth**	**Height**	**Balance factor**
A			
AA			
AB			
AC			
AAA			
ABA			
ABC			

Task 3 (6 marks)

Write the Pre-Order, In-Order and Post-Order traversal of the tree below in your report.