Revision Lab 1

Task 1 DoublyLinkedNode class

Check out DoublyLinkedNode class from Lab 6 and familiarize yourself with how to implement the functions. Fill in the missing parts of the DoubleLinkedNode.h below from memory.

DoubleLinkedNode.h

```
template<class DataType>
class DoublyLinkedNode
{
public:
	typedef DoublyLinkedNode<DataType> Node;

private:
	DataType fValue;
	Node* fNext;
	Node* fPrevious;

	DoublyLinkedNode()
	{
		fValue = DataType();
		fNext = &NIL;
		fPrevious = &NIL;
	}

public:
	static Node NIL;

	DoublyLinkedNode(const DataType& aValue);

	void prepend(Node& aNode);
	void append(Node& aNode);
	void remove();

	const DataType getValue() const;
	const Node* getNext() const;
	const Node* getPrevious() const;
};

template<class DataType>
DoublyLinkedNode<DataType> DoublyLinkedNode<DataType>::NIL;

template<class DataType>
DoublyLinkedNode<DataType>::DoublyLinkedNode(const DataType& aValue)
{
	///////(fill in what is missing)
}

template<class DataType>
void DoublyLinkedNode<DataType>::prepend(Node& aNode)
{
	///////(fill in what is missing)
}

template<class DataType>
void DoublyLinkedNode<DataType>::append(Node& aNode)
```

```
{
	///////(fill in what is missing)
}

template<class DataType>
void DoublyLinkedNode<DataType>::remove()
{
	if (fNext == &NIL)
	{
		///////(fill in what is missing)
	}
	else if (fPrevious == &NIL)
	{
		///////(fill in what is missing)
	}
	else
	{
		///////(fill in what is missing)
	}

	//delete this;
}

template<class DataType>
const DataType DoublyLinkedNode<DataType>::getValue () const {
	///////(fill in what is missing)
}

template<class DataType>
const DoublyLinkedNode<DataType> *DoublyLinkedNode<DataType>::getNext() const {
	///////(fill in what is missing)
}

template<class DataType>
const DoublyLinkedNode<DataType> *DoublyLinkedNode<DataType>::getPrevious() const {
	///////(fill in what is missing)

}
```

Task 2 Copy Control Elements

Go through Lab 7 and learn how to use and identify copy constructor, assignment operator and clone(). Revise on these copy control elements in Lecture 7 Slides 28 to 40.

Task 3 List, Stack and Queue

Stack is a **LIFO** data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called **push** operation and removal operation is called **pop** operation.

Queue on the other hand is a **FIFO** data structure. FIFO stands for First-in-first-out. The element that is placed first will be accessed first.

Revise on List and Stack in Lecture 6 Slides 7 to 21 and revise on Queue in Lecture 7 Slides 2 to 7.