CMPSCI 187 Discussion #9: Implementing Double-Ended Queues
Individual Handout

David Mix Barrington
31 October 2012

Here is the `Deque` interface from Discussion #7:

```
public interface Deque<T> {
            public void addToFront (T element);
            public T removeFront ( ) throws EmptyCollectionException;
            public T first ( ) throws EmptyCollectionException;
            public void addToRear (T element);
            public T removeRear ( ) throws EmptyCollectionException;
            public T last ( ) throwsEmptyCollectionException;
            public boolean isEmpty ( );
            public int size ( );}
```

And here are the `LLNode` and `DLLNode` generic classes from DJW:

```
public class LLNode<T> {
   private LLNode link;
   private T info;
   public LLNode (T info) {this.info = info; link = null;}
   public void setInfo (T info) {this.info = info;}
   public T getInfo ( ) {return info;}
   public void setLink (LLNode link) {this.link = link;}
   public LLNode getLink ( ) {return link;}}

public class DLLNode<T> extends LLNode<T> {
   private DLLNode<T> back;
   public DLLNode (T info) {super(info); back = null;}
   public void setBack (DLLNode<T> back) {this.back = back;}
   public DLLNode<T> getBack( ) {return back;}}
```

Your task is a simple one – write a generic class `DLLDeque<T>` that implements the `Deque<T>` interface using doubly linked lists of `DLLNode<T>` objects. A `DLLDeque` object will have three instance fields – pointers to the first and last nodes, and an *int* storing the number of elements in the deque. You may find it easier to use header and trailer nodes, so that your doubly linked list always has at least two nodes.

There are eight methods to write, and there is a symmetry between the forward and backward directions, so that each of the first three methods has a matching method that can be obtained from it by reversing all the directions.