```java
/**
 * JUnit test class.  Use these tests as models for your own.
 */

import org.junit.*;


import org.junit.rules.Timeout;

import static org.junit.Assert.*;

import hwk2.LinkedList;
import sun.awt.image.ImageWatched;


public class LinkedListTest {
    @Rule
    // a test will fail if it takes longer than 1/10 of a second to run
    public Timeout timeout = Timeout.millis(100);


    @Test
    //Remove empty LinkedList.
    //Return null, should have 0 nodes, the content doesn't change
    public void testRemoveHeadEmpty(){
        LinkedList ll = new LinkedList();
        assertNull(ll.removeHead());
        assertEquals(0,ll.getLength());
        assertEquals("()",ll.toString());
    }
    @Test
    //Remove LinkedList with one element. Return the only element after
removing
    //Should have 0 nodes, the content will change
    public void testRemoveHeadOne(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");

        String expected = ll.removeHead();
        assertEquals(0,ll.getLength());
        assertEquals("A",expected);
        assertEquals("()",ll.toString());

    }
    @Test
    //Remove LinkedList with more than one element
    //The length should decrement by 1. Return the removed element.
    // Content will change
    public void testRemoveHeadMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("B");
        ll.insertAtTail("C");
```

```
        ll.insertAtTail("D");
        String expected = ll.removeHead();
        assertEquals(3,ll.getLength());
        assertEquals("A",expected);
        assertEquals("(B,C,D)",ll.toString());




    }

    @Test
    //Remove the first element in a multiple element LinkedList even though
it is identical to the others.
    //Should remove the exact element in the exact position. Length will
decrement by 1.
    //Should not alter the values besides removing from the LinkedList
    public void removeHeadSameMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("A");
        ll.insertAtTail("A");
        ll.insertAtTail("A");
        ll.insertAtTail("A");
        assertEquals("A",ll.removeHead());
        assertEquals(4,ll.getLength());
        assertEquals("(A,A,A,A)",ll.toString());

    }

    @Test
    //Remove head of a LinkedList that has two identical elements and after
removing there will only be one left
    //The length should decrement by 1. Content will change
    public void testRemoveHeadSameTwice(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("A");
        assertEquals("A",ll.removeHead());
        assertEquals(1,ll.getLength());
        assertEquals("(A)",ll.toString());
    }
    @Test
    //Insert an empty LinkedList. Then it became the first element of
LinkedList
    //The length should increment by 1, content will change
    public void testInsertTailEmpty(){
        LinkedList ll = new LinkedList();
        ll.insertAtTail("A");
        assertEquals(1,ll.getLength());
        assertEquals("(A)",ll.toString());

    }
```

```
    @Test
    //Insert LinkedList with more than one elements
    //The length should increment by 1
    // content will change as the inserted element will be added at the end
    public void testInsertTailMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("C");
        ll.insertAtHead("B");
        ll.insertAtHead("A");
        ll.insertAtTail("Z");
        assertEquals(4,ll.getLength());
        assertEquals("(A,B,C,Z)",ll.toString());


    }
    @Test
    //Insert to one element LinkedList
    //The length should increment by 1
    //content will change as the inserted element will be added at the end
    public void testInsertTailOne(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("Z");
        assertEquals(2,ll.getLength());
        assertEquals("(A,Z)",ll.toString());

    }

    @Test
    //Insert an identical element to a one-element LinkedList.
    // The LinkedList will have identical elements instead of unable to
insert.
    //Length will increment by 1

    public void testInsertTailSameOne(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("A");
        assertEquals(2,ll.getLength());
        assertEquals("(A,A)",ll.toString());


    }

    @Test
    //Insert tail of a multiple identical element LinkedList. Length and
content should change
    public void testInsertTailSameMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("F");
        ll.insertAtHead("F");
        ll.insertAtHead("F");
        ll.insertAtHead("F");
```

```
        ll.insertAtHead("F");
        ll.insertAtTail("F");
        assertEquals(6,ll.getLength());
        assertEquals("(F,F,F,F,F,F)",ll.toString());
    }
    @Test
    //IndexOf empty LinkedList. Return -1. Should not change LinkedList
content nor the length
    public void testIndexOfEmpty(){
        LinkedList ll = new LinkedList();
        assertEquals(-1,ll.indexOf("A"));
        assertEquals("()",ll.toString());
        assertEquals(0,ll.getLength());

    }
    @Test
    //IndexOf invalid data of one element LinkedList. Return -1
    //Should not change LinkedList content nor the length
    public void testIndexOfInvalidOne(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        assertEquals(-1,ll.indexOf("B"));
        assertEquals(1,ll.getLength());
        assertEquals("(A)",ll.toString());



    }

    @Test
    //IndexOf invalid data of more than one element LinkedList. Return -1
    //Should not change LinkedList content nor the length
    public void testIndexOfInvalidMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("B");
        ll.insertAtTail("C");
        ll.insertAtTail("D");
        assertEquals(-1,ll.indexOf("E"));
        assertEquals(4,ll.getLength());
        assertEquals("(A,B,C,D)",ll.toString());



    }
    @Test
    //IndexOf one element LinkedList. Return the index of that data (0)
    //Should not change LinkedList content nor the length
    public void testIndexOfOne(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        assertEquals(0,ll.indexOf("A"));
        assertEquals(1,ll.getLength());
```

```
        assertEquals("(A)",ll.toString());



    }
    @Test
    //IndexOf more than one element LinkedList
    //Should return the corresponding indexes when given valid data
    //Should not change LinkedList content nor the length
    public void testIndexOfMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("B");
        ll.insertAtTail("C");
        ll.insertAtTail("D");
        assertEquals(0,ll.indexOf("A"));
        assertEquals(1,ll.indexOf("B"));
        assertEquals(2,ll.indexOf("C"));
        assertEquals(3,ll.indexOf("D"));
        assertEquals(4,ll.getLength());
        assertEquals("(A,B,C,D)",ll.toString());
    }
    @Test
    //IndexOf identical data that appear in the LinkedList more than once.
    //Return the first occurrence of that data
    //Should not change LinkedList content nor the length
    public void testIndexOfSameMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("C");
        ll.insertAtTail("C");
        ll.insertAtTail("C");
        ll.insertAtTail("C");
        ll.insertAtTail("C");
        assertEquals(0,ll.indexOf("C"));
        assertEquals(5,ll.getLength());
        assertEquals("(C,C,C,C,C)",ll.toString());



    }

    @Test
    //Get index of a LinkedList that has two identical elements. Should get
the correct index (0).
    //Should not change content nor length
    public void testIndexOfSameTwice(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("A");
        assertEquals(0,ll.indexOf("A"));
        assertEquals(2,ll.getLength());
        assertEquals("(A,A)",ll.toString());
    }
```

```
    @Test
    //Test insertAtHead to an empty LinkedList. The inserted element will be
the first element in the LinkedList
    //Content will change, the length will increment by 1
    public void testInsertHeadEmpty(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        assertEquals("(A)",ll.toString());
        assertEquals(1,ll.getLength());

    }
    @Test
    //Test insertAtHead to a LinkedList with one element. The length will
increment by 1. Content will change
    public void testInsertHeadOne(){
        LinkedList ll = new LinkedList();
        ll.insertAtTail("A");
        ll.insertAtHead("D");
        assertEquals(2,ll.getLength());
        assertEquals("(D,A)",ll.toString());

    }

    @Test
    //Test insertAtHead to a LinkedList with more than one elements. The
content and length will change
    public void testInsertHeadMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtTail("S");
        ll.insertAtTail("D");
        ll.insertAtTail("A");
        ll.insertAtHead("B");
        assertEquals(4,ll.getLength());
        assertEquals("(B,S,D,A)",ll.toString());

    }

    @Test
    //Test insertAtHead to a one element LinkedList by inserting the
identical element
    //The content and length will change.
    //Should insert at the beginning instead of unable to insert
    public void testInsertHeadSameOne(){
        LinkedList ll = new LinkedList();
        ll.insertAtTail("D");
        ll.insertAtHead("D");
        assertEquals(2,ll.getLength());
        assertEquals("(D,D)",ll.toString());

    }

    @Test
```

```
    //Test insertAtHead of a LinkedList with multiple identical elements.
Content and length should change
    public void testInsertHeadSameMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtTail("D");
        ll.insertAtTail("D");
        ll.insertAtTail("D");
        ll.insertAtTail("D");
        ll.insertAtTail("D");
        ll.insertAtHead("D");
        assertEquals(6,ll.getLength());
        assertEquals("(D,D,D,D,D,D)",ll.toString());


    }

    @Test
    //Test isEmpty of an empty LinkedList. Return true
    public void testIsEmptyTrue(){
        LinkedList ll = new LinkedList();
        assertTrue(ll.isEmpty());

    }

    @Test
    //Test isEmpty of a non-empty LinkedList. Should return false
    public void testIsEmptyFalse(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("F");
        ll.insertAtTail("D");
        assertFalse(ll.isEmpty());

    }

    @Test
    //Test getLength of an empty LinkedList. Return 0
    public void testGetLengthEmpty(){
        LinkedList ll = new LinkedList();
        assertEquals(0,ll.getLength());
    }

    @Test
    //Test getLength of a non-empty LinkedList. Return the length
    public void testGetLengthNonEmpty(){
        LinkedList ll = new LinkedList();
        ll.insertAtHead("A");
        ll.insertAtTail("R");
        assertEquals(2,ll.getLength());
    }

    @Test
    //Test toString of empty LinkedList.
    public void testToStringEmpty(){
```

```
        LinkedList ll = new LinkedList();
        assertEquals("()",ll.toString());

    }

    @Test
    //Test toString of multiple elements LinkedList
    public void testToStringMultiple(){
        LinkedList ll = new LinkedList();
        ll.insertAtTail("A");
        ll.insertAtTail("D");
        ll.insertAtHead("F");
        assertEquals("(F,A,D)",ll.toString());
    }

    @Test
    //Test toString of one element LinkedList
    public void testToStringOne(){
        LinkedList ll = new LinkedList();

        ll.insertAtHead("F");
        assertEquals("(F)",ll.toString());
    }

}
```

```java
package hwk2;

/**
 * ListNode is a building block for a linked list of data items
 *
 * @author C. Fernandes
 * @version 9/30/2017
 */
public class ListNode
{
    public String data;
    public ListNode next;

    /** Non-default constructor
     *
     * @param value a reservation you want stored in this node
     */
    public ListNode(String value)
    {
        data = value;
        next = null;
    }

    /**
     * returns data as a printable string
     */
    public String toString()
    {
        return data;
    }
}
```

```java
package hwk2;

/**
 * Linked List is a collection of data nodes.  All methods here relate to
 * how one can manipulate those nodes.
 *
 * @author Emma Vu
 * @version 5/4/2020
 *
 * I affirm that I have carried out the attached academic endeavors with full
 academic honesty, in
 * accordance with the Union College Honor Code and the course syllabus
 */
public class LinkedList
{
    private int length;          // number of nodes
    private ListNode firstNode;  // pointer to first node

    public LinkedList()
    {
        length=0;
        firstNode=null;
    }


    /** insert new String at linked list's head
     *
     * @param newData the String to be inserted
     */
    public void insertAtHead(String newData)
    {
        ListNode newnode = new ListNode(newData);
        if (isEmpty())
        {
            firstNode=newnode;
        }
        else
        {
            newnode.next=firstNode;
            firstNode=newnode;
        }
        length++;
    }

    /** remove and return data at the head of the list
     *
     *  @return the String the deleted node contains.  Returns null if list
empty.
     */
    public String removeHead()
    {
        if(isEmpty()){
            return null;
```

```
        }
        else{
          String remove = firstNode.toString();
          firstNode = firstNode.next;
          length--;
          return remove;
        }

    }

    /** insert data at end of list
     *
     * @param newData new String to be inserted
     */
    public void insertAtTail(String newData)
    {
        ListNode insertNode = new ListNode(newData);
        if(isEmpty()){
            firstNode = insertNode;
        }
        else{

            ListNode runner = firstNode;

            while(runner.next!=null){
                runner = runner.next;
            }
            runner.next = insertNode;

        }
        length++;
    }

    /**
     * search for first occurrence of value and return index where found
     *
     * @param value string to search for
     * @return index where string occurs (first node is index 0).  Return -1
if value not found.
     */
    public int indexOf(String value)
    {   int index = 0;

        ListNode runner = firstNode;
        while(runner!= null){
            if (runner.data.equals(value)){
                return index;
            }
            else{
                runner = runner.next;
                index ++;
            }
        }
```

```
        return -1;

    }

    /**
     *  @return return linked list as printable string
     */
    public String toString()
    {
        String toReturn="(";
        ListNode runner=firstNode;
        while (runner!=null)
        {
            toReturn = toReturn + runner;  //call node's toString
automatically
            runner=runner.next;
            if (runner!=null)
            {
                toReturn = toReturn + ",";
            }
        }
        toReturn = toReturn + ")";
        return toReturn;
    }

    /**
     *
     * @return length of LL
     */
    public int getLength() {return length;}

    /**
     *
     * @return true if LL empty or false if not
     */
    public boolean isEmpty() {return getLength()==0;}
}
```