**Question 0.**  [2 MARKS]
Complete the “identification section” at the top of page 1, then write your student number **legibly** at the bottom of every page of this test except page 1 (where indicated).

**Question 1.**  [15 MARKS]
Consider running the main method in the following class.

```java
public class DLNode {

    private static int num = 0;
    private int value;
    private DLNode next;
    private DLNode prev;

    public DLNode(DLNode prev, int value, DLNode next) {
        num++;
        this.value = value;
        this.prev = prev;
        this.next = next;
        prev = next;
        next = prev;
    }

    public static void moonwalk(DLNode n, int c) {
        if (c > 0)
            moonwalk(n.next, c - 1);
        System.out.print("" + n.value);
    }

    public static void main(String[] args) {
        DLNode root = new DLNode(null, 99, null);
        root.next = new DLNode(root, 66, null);
        root.next.next = new DLNode(root.next, 77, root);
        root.prev = root.next.next;

        // Line number 5.
        System.out.println(num);
        moonwalk(root, 3);
        System.out.println();
    }
}
```

**Part (a)**  [5 MARKS]
What does the program above print when it is compiled and run? This is not a trick question: the program does compile and run without error. (Hint: You may wish to do the next part of this question first.)
Question 1.  (continued)

Part (b)  [10 marks]

Sketch the memory model for the program on the previous page at the point when the execution reaches "// Line number 5" of the main method. To keep the sketch small, only draw the items for the given DLNode class in the object space. Include the run-time stack, the object space, and the static space in your sketch.
Question 2. [13 MARKS]
Complete the method deleteRange below according to the contract specified by all the comments in the LList class. Include appropriate internal comments.

```java
// Nodes for our linked list.
public class LNode {
    int value;
    LNode next;
}

// Exception class for our linked list.
public class ListUnderflowException extends Exception {}

// A simple linked list class.
public class LList {
    /* Representation invariant: Either
     * a) head == null and size == 0, or
     * b) head != null and size == the number of elements of this linked list.
     */
    private LNode head;
    private int size;

    /* Assume that 'insert' and other methods are here... */

    // Delete the n consecutive elements starting at position k of this linked list.
    // (Note: position 0 corresponds to the head of the list.)
    // Precondition: k >= 0 and n >= 0
    // Throws ListUnderflowException if the list contains less than k+n elements.
    public void deleteRange(int k, int n) throws ListUnderflowException {
        if (size < k + n)
            throw new ListUnderflowException();

        if (k == 0) {
            // The k-th element is the head; move it forward n positions.
            while (n > 0) {
                head = head.next;
                size--;
                n--;
            }
        } else {
            // Find the predecessor of the k-th element.
            LNode pred = head;
            for (int i = 0; i < k - 2; i++)
                pred = pred.next;
            // Delete n consecutive elements starting with the k-th.
            while (n > 0) {
                pred.next = pred.next.next;
                size--;
                n--;
            }
        }
    }
```


```java
}
} // end of deleteRange(int,int)
} // end of class LList
```
Question 3.  [10 marks]
Consider the classes below that implement a “Ternary Search Tree”. A Ternary Search Tree is a labelled tree with a branching factor of 3 that satisfies the following property: for every node n in the tree,

- every key in the left subtree of n is less than the key at n,
- every key in the middle subtree of n is equal to the key at n,
- every key in the right subtree of n is greater than the key at n.

(Obviously, duplicate keys are allowed in a Ternary Search Tree.)
Write the body of method contains in class LinkedSimpleTST below so that it meets its specification, without using recursion. Include appropriate internal comments.

class TSTNode {
    Comparable key;
    TSTNode left, middle, right;

    TSTNode(Comparable key) { this.key = key; }
}

public class LinkedSimpleTST {
    // The root of this TST.
    private TSTNode root;

    // Assume that methods ‘insert’ and ‘delete’ are here...

    // Return true if this tree contains key, false otherwise.
    public boolean contains(Comparable key) {

        // Starting at the root, traverse the tree looking for ‘key’ until
        // it is found, or until a null reference is reached.
        TSTNode current = root;
        while (current != null && key.compareTo(current.key) != 0) {
            if (key.compareTo(current.key) < 0) {
                current = current.left;
            } else { // key > current.key
                current = current.right;
            }
        }

        // We know ‘key’ was found iff current is not null.
        return (current != null);
    }

} // end of contains(Comparable)
} // end of class LinkedSimpleTST

Total Marks = 40