**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.** [10 MARKS]
Complete the method `deleteSecondLast` 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 second-last element of this linked list
    // (i.e., the element that comes just before the last one).
    // Throws ListUnderflowException if the list contains less than two elements.
    public void deleteSecondLast() throws ListUnderflowException {
        if (size < 2)
            throw new ListUnderflowException();

        if (size == 2) {
            // The second-last element is the head; delete it.
            head = head.next;
        } else { // size > 2
            // Find the predecessor of the second-last element.
            LNode pred = head;
            for (int i = 0; i < size - 3; i++)
                pred = pred.next;
            // Delete the second-last element.
            pred.next = pred.next.next;
        }

        // Update the size of this list.
        size--;
    }
}
} // end of deleteSecondLast()
} // end of class LList
Question 2.  [15 MARKS]
Consider running the main method in the following class.

```java
public class GphNode {
    private static int m = 0;
    private int value;
    private GphNode edgeA;
    private GphNode edgeB;

    public GphNode(int value, GphNode edgeA, GphNode edgeB) {
        m = value;
        this.value = value;
        this.edgeA = edgeA;
        this.edgeB = edgeB;
        edgeA = edgeB;
        edgeB = edgeA;
    }

    public static void hopAlong(GphNode n, int c) {
        System.out.print(" "+n.value);
        if (c > 0)
            hopAlong(n.edgeA.edgeB.edgeA.edgeA, c - 1);
    }

    public static void main(String[] args) {
        GphNode start = new GphNode(9, null, null);
        start.edgeB = new GphNode(6, start, null);
        start.edgeB.edgeB = new GphNode(5, start.edgeB, start);
        start.edgeA = start.edgeB.edgeB;

        // Line number 5.
        System.out.println(m);
        hopAlong(start, 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 2. (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 GphNode class in the object space. Include the run-time stack, the object space, and the static space in your sketch.
Question 3.  [13 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 countOccurrences in class LinkedSimpleTST below so that it meets its specification, without using recursion. Include appropriate internal comments.

```java
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 the number of times that key appears in this tree
    // (return 0 if key does not appear at all in this tree).
    public int countOccurrences(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;
            }
        }

        // Count the number of occurrences of key.
        int count = 0;
        while (current != null) {
            count++;
            current = current.middle;
        }

        return count;
    }
}
```
} // end of countOccurrences(Comparable)
} // end of class LinkedSimpleTST

Total Marks = 40