University of Colombo
Faculty of Technology
Department of Information and Communication Technology
IC 2306 – Programming II
Level 2 – Semester II

Lab Sheet 03

01. The program below shows java implementation to demonstrate insertion, deletion & search operations on Singly Linked List.
    Observe the program and fill in the blanks with suitable code lines in LinkedList.java class and Main.java class in order to get the expected output.

Output

```
Yes
80   |   30   |   10   |   70
```

LinkedList.java

```
public class LinkedList {

      class Node{
            public int data;
//Reference to the next Node
            public (a)_ _ _ _ _ _;

//Parameterized constructor
            public Node(int item)
            {
                  (b)_ _ _ _ _ _;
                  (c)_ _ _ _ _ _;
            }

//Default constructor
            public (d)_ _ _ _ _ _
            {
                  data = 0;
                  next = null;
            }
      }

       Node head;

      public void insertAtLast(int data) {
//Create a new node. Set value of the node to data and next pointer to null.
            (e) _ _ _ _ _ _ _ _ _ _ _ _;
            (f) _ _ _ _ _ _ _ _ _ _ _ _;
            (g) _ _ _ _ _ _ _ _ _ _ _ _;
```

```
//Check if the list is empty. Point head to the new node
            if((h) _ _ _ _ _ _ _ _ _ _ _ _) {
                  (i) _ _ _ _ _ _ _ _ _ _ _ _;
            }
            else
            {
//Set head pointer to the currentNode
                  Node currentNode = head;
//Traverse through the Linked List.
                  while(currentNode.next!=null)
                  {
                        currentNode =currentNode.next;
//Point next pointer of the current node to the new Node
                         (j) _ _ _ _ _ _ _ _ _ _ _ _;
            }
      }



      public void insertAtStart(int data)
      {
//Create a new Node. Set value of the node to data.
            (k) _ _ _ _ _ _ _ _ _ _ _ _;
            (l) _ _ _ _ _ _ _ _ _ _ _ _;

//Link the new node to the previous node so that new node become head node;
            (m) _ _ _ _ _ _ _ _ _ _ _ _;
            (n) _ _ _ _ _ _ _ _ _ _ _ _;
      }


      public void insertAtMiddle(int index, int data)
      {

//Create a new Node. Set value of the node to data.
            (p) _ _ _ _ _ _ _ _ _ _ _ _;
            (q) _ _ _ _ _ _ _ _ _ _ _ _;

//Check if the node to be added in the beginning of the list
            if(index==0)
            {
                  insertAtStart(data);
            }
            else
            {
//Set head pointer to the currentNode
            Node currentNode = head;
```

```
//Traverse to node just before the required position of new node
              for(int i=0; i<index; i++)
              {
                     currentNode=currentNode.next;
              }

//Link the current node to new node
              (r) _ _ _ _ _ _ _ _ _ _ _ _;

//Link the new node to the next node
              (s) _ _ _ _ _ _ _ _ _ _ _ _;
              }
       }

       public void delete(int value)
       {
//Store head node
              Node deleteNode = head;
              Node currentNode = null;

//Search for the key to be deleted
              while(deleteNode!=null && deleteNode.data!=value) {
                     currentNode = deleteNode;
                     deleteNode = deleteNode.next;
              }
//Unlink the node from the link list
              (t) _ _ _ _ _ _ _ _ _ _ _ _ _ _;
      }


// Checks whether the value data is present in linked list
       public boolean search(Node head, int data)
       {
              Node currentNode = head;
              while(currentNode!=null)
              {
//Traverse through the Linked List
//Return true if the value of the current Node is equal to the given data else
return false
                     if((u) _ _ _ _ _ _ _ _ _ _ _ _)
                            return true;
                         (v) _ _ _ _ _ _ _ _ _ _ _ _;
                     }
                     return false;
              }
//Display the Linked List
       public void displayList() {
              Node node = head;
```

```
            while(node.next!=null)
            {
                  System.out.print(node.data + " | ");
                  node = node.next;
            }
            System.out.print(node.data);
      }
}
```

Main.java

```
public class Main {
      public static void main(String[] args) {

            LinkedList list = new LinkedList();
            list.insertAtLast((i)_ _ _);
            list.insertAtStart((ii) _ _ _);
            list. (iii) _ _ _ _ _ _ _ _ _ _;
            list.insertAtMiddle(2, 90);
            list.delete((iv) _ _ _);



            if(list.search(list.head, 30))
                  System.out.println("Yes");
            else
                  System.out.println("No");


            list.displayList();
      }
}
```

02. The program below shows java implementation of a doubly linked list with the insertion of new nodes at the end of the list.

Observe the given java classes and fill in the blanks with suitable code lines in order to produce the expected output.

Output:

```
Nodes of doubly linked list:

10 20 30 40
```

DoublyLinkedList.java

```
public class DoublyLinkedList {
	class Node{
		(i) _ _ _ _ _ _ _ _ _;
		(ii) _ _ _ _ _ _ _ _ _;
		(iii) _ _ _ _ _ _ _ _ _;

		public Node(int item)
		{
			data = item;
			next = null;
			prev = null;
		}

		public Node()
		{
			data = 0;
			next = null;
			prev = null;
		}
	}

	//Set head and tail to null
	(iv) _ _ _ _ _ _ _ _ _ _ _;


//Function to insert node in the list
	public void insertNode(int value)
	{
		//create a new node
		(v) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _;
```

```
            //If list is empty, head and tail points to newNode
            if((vi) _ _ _ _ _ _ _)
            {
                  head = tail = newNode;
                  //Set previous pointer of the head node to null
                  (vii) _ _ _ _ _ _ _;

                  //Set previous pointer of the head node to null
                  (viii) _ _ _ _ _ _ _;
            }
            else
            {
                  //Add new node to the end of the list. Set next pointer of the
tail node to newNode
                  (ix) _ _ _ _ _ _ _ _ _ _;
                  //Set previous pointer of the new Node of the  tail
                  newNode.prev = tail;
                  //Set newNode to new tail
                  (x) _ _ _ _ _ _ _ _ _ _;
                  //Set next pointer of the tail to null
                  tail.next=null;
            }

      }


//Function to print all the nodes of doubly linked list
      public void printList()
      {
            //Point head to the node current.
            Node currentNode = head;
            if(head==null)
            {
                  return;
            }
            System.out.println("Nodes of the Doubly Linked List: ");
            //Traverse through the Doubly link list and print each node
            while ((xi)_ _ _ _ _ _ _)
            {
                  System.out.println(" " + (xii) _ _ _ _ _ _ _);
                  (xiii) _ _ _ _ _ _ _ _ _ _ _ _ _ _;
            }
      }
}
```

Main.java

```
public class Main {
	public static void main(String[] args)
	{
		//Create a Doubly Linked List Object
		(xiv) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _;
		//Add nodes to the list
		dbl_List.insertNode((xv)_ _ _);
		dbl_List.insertNode((xvi)_ _ _);
		dbl_List.(xvii)_ _ _ _ _ _ _ _);
		(xviii)_ _ _ _ _ _ _ _ _ _ _;
		// Print the nodes of doubly linked list
		dbl_List.printList();
	}
}
```

03. The program below shows java implementation of a Circular Linked List with the insertion and search operations.
Observe the given java classes and fill in the blanks with suitable code lines.

```
public class CircularLinkedList {
	class Node
	{
	   int data;
	   Node next;

	   public Node(int data)
	   {
	       this.data = data;
	   }
	}

//Initially head and tail is set to null.
	(a)_ _ _ _ _ _ _ _ _;
	(b)_ _ _ _ _ _ _ _ _;

//Function to insertNode to the list
	public void insertNode(int value)
	{
```

```
            Node newNode = new Node(value);

//Check if head node is null. Set new node as head. Else point existing tail to the
new node.
            if(head==null) {
                  (c)_ _ _ _ _ _ _ _ _;
            }
            else {
                  tail.next=newNode;
            }
//Link new node and the existing node
            (d)_ _ _ _ _ _ _ _ _;
            (e)_ _ _ _ _ _ _ _ _;
      }

      public boolean searchNode(Node head, int value) {

//Set the head node as the currentNode
            (f)_ _ _ _ _ _ _ _ _ _ _;

//Traverse through the entire list
//Return false if the list is empty
            if(head==null) {
                  (g)_ _ _ _ _ _ _ _;
            }
            else
            {
                  do
                  {
//Return true if the value of the currentNode is the value to search
                        if((h)_ _ _ _ _ _ _ _ _ _ _ _ _) {
                              return true;
                        }
                        currentNode = currentNode.next;

                  }while (currentNode!=head);
                  return false;
            }
      }
```