Madan Lal (053-18-0005)

Lab 02

Task 01

```java
class Node {

        String name;
        Node prev, next;

        Node (String name)
        {
            this.prev = null;
            this.next = null;
            this.name = name;
        }
}

public class DoubleLinkedList {

    Node head;

    // Add node with name in beginning of linkedlist, name as param
    public void insertAtBeginning(String name)
    {
        Node newNode = new Node(name);
        insertAtBeginning(newNode);
    }

    // Add node with name in beginning of linkedlist, node as param
    public void insertAtBeginning(Node newNode)
    {

        if(head == null) {
            head = newNode;
        }
        else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }

    }
    // Add node in end of linedlist, name as param
    public void insertAtEnd(String name)
    {
        Node newNode = new Node(name);
        insertAtEnd(newNode);
    }
    // Add node in end of linedlist, node as param
    public void insertAtEnd(Node newNode)
    {
        Node currNode = head;
        while(currNode.next != null) {
            currNode = currNode.next;
        }
        currNode.next = newNode;
        newNode.prev = currNode;

    }
    // Add node after name which is provided as param , name and node as params
    public void insertAftername(String name, Node newNode)
    {
        Node currNode = head;
        Node prevNode = null;

        while(currNode.name != name) {
            prevNode = currNode;
            currNode = currNode.next;
        }

        newNode.next = currNode.next;
        newNode.prev = currNode;
        currNode.next = newNode;

    }
    // Add node before name which is provided as param , name and node as
params
    public void insertBeforename(String name, Node newNode)
    {
        Node currNode = head;
        Node prevNode = null;

        while(currNode.name != name) {
            prevNode = currNode;
            currNode = currNode.next;
        }
        newNode.next = currNode;
        newNode.prev = prevNode;
        prevNode.next = newNode;
        currNode.prev = newNode;
    }

    // Print all the nodes in linkedlist, make sure it works on circular double
linkedlist
    public void printAll()
    {
        Node currNode = head;

        while(currNode != null) {

            System.out.print(currNode.name + " -> ");
            currNode = currNode.next;
        }
        System.out.print("NULL");
    }
    // Test the class
    public static void main(String[] args) {

        DoubleLinkedList list = new DoubleLinkedList();

        list.insertAtBeginning("This");
        list.insertAtEnd("Apple");

        Node newNode = new Node("is");
        list.insertAftername("This",newNode);

        Node newNode2 = new Node("an");
        list.insertBeforename("Apple", newNode2);

        list.printAll();
        list.makeCircular();

    }

}
```

Task 02

```
public class SinglyLinkedList_tail{
    // initialize Node as public...
    Node head, tail;

    // established Node Class
    class Node {

        String data;
        Node next;

        Node(String data) {
            this.data = data;
            this.next = null;
        }
    }

    // construct function for adding node at first/middle/last...
    public void addNode(String data) {

        Node newNode = new Node(data);

        if(head == null) {

            head = tail = newNode;
            return;
        }

        else {

            Node currNode = head;

            while(currNode.next != null) {

                currNode = currNode.next;
            }

            currNode.next = newNode;
            tail = newNode;
            }
    }

    // isEmpty method return true if linkedlist is empty else false...
    public void isEmpty() {
        if(head == null)
            System.out.println("true");
        else
```

```
            System.out.println("false");
    }

    // size method will return size of the linkedlist...
    public void size() {

        int count = 0;

        if(head == null) {

            count = 0;

        }
        else {

            Node currNode = head;

            while(currNode != null) {

                count++;
                currNode = currNode.next;
            }

        }

        System.out.println(count);
    }

    // removeNode method will remove node from linkedlist...
    public void removeNode(String data) {

        if(head == null) {

            System.out.println("No Nodes available!");

        }
        else if(head.data == data) {

            head = head.next;
        }
        else {

            Node currNode = head;
            Node prevNode = null;
            while(currNode.next != null) {

                    if(currNode.data == data) {

                        prevNode.next = currNode.next;
                        return;
                    }
                prevNode = currNode;
                currNode = currNode.next;
            }


        }
    }


    // construct method for printing nodes...
    public void Print() {

        if(head == null) {

            System.out.println("Linkedlist is empty!");

        }
        else {

            Node currNode = head;

            while(currNode != null) {

                System.out.print(currNode.data + " -> ");
                currNode = currNode.next;
            }
            System.out.println("NULL");
        }
    }

    public static void main(String[] args) {

        SinglyLinkedList_tail myMethod = new SinglyLinkedList_tail();

        myMethod.addNode("A");
        myMethod.addNode("B");
        myMethod.addNode("C");
        myMethod.Print();
        myMethod.isEmpty();
        myMethod.size();
        myMethod.removeNode("B");
        System.out.println("After deleting node...");
        myMethod.Print();

    }
}
```

Task 03

```
public class LinkedList_Cycle{

    // initialize Node as public...
    Node head;

    // established Node Class
    class Node {

        String data;
        Node next;

        Node(String data) {
            this.data = data;
            this.next = null;
        }
    }

    // construct function for adding node at first/middle/last...
    public void addNode(String data) {

        Node newNode = new Node(data);

        if(head == null) {

            head = newNode;
            return;
        }

        else {

            Node currNode = head;

            while(currNode.next != null) {
```

```
            currNode = currNode.next;
        }

        currNode.next = newNode;
        }
}

// ----------------
public void isCycle() {

    boolean cycle = false;
    Node currNode = head;
    Node pointer = head.next;

        while(currNode != null) {

            while(pointer != null)
            {
                if(currNode.data == pointer.data)
                {
                    cycle = true;
                }

                pointer = pointer.next;
            }

            currNode = currNode.next;

        }
        if(cycle) {
            System.out.println("Cycle...");
        }
        else {
            System.out.println("No Cycle...");
        }
}


// construct method for printing nodes...
public void Print() {

    if(head == null) {

        System.out.println("Linkedlist is empty!");
        }
        else {

            Node currNode = head;

            while(currNode != null) {

                System.out.print(currNode.data + " -> ");
                currNode = currNode.next;
            }
            System.out.println("NULL");
        }
    }

    public static void main(String[] args) {

        LinkedList_Cycle myMethod = new LinkedList_Cycle();

        myMethod.addNode("A");
        myMethod.addNode("B");
        myMethod.addNode("C");
        myMethod.Print();
        myMethod.isCycle();

    }
}
```