NAME: LINDA KELLEN AYEBALE

STUDENT NO: 2000700059

REG NO.: 20/U/0059

Implementation of the algorithms in the lecture slides

```
public class linkedlist{

    public class Node{
        int data;
        Node next;

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

    public Node head = null;
    public Node tail = null;

    public void addNode(int data){
        Node node = new Node(data);

        if(head==null){
            head = node;
            tail = node;
        }
        else{
            tail.next = node;
            tail = node;
        }
    }

    public void show(){
        Node current = head;

        if(head== null){
            System.out.println("The list is empty.");
        }
        else{
            while(current!=null){
                System.out.printf("%d%n",current.data);
                current = current.next;
            }
```

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

    public void insertAtStart(int data){
        Node node = new Node(data);

        node.data = data;
        node.next = head;
        head = node;
    }

    public void InsertAfter(int data){
        Node node = new Node(data);
        Node current = head;
        int Key = 45;

        if(head == null){
            System.out.println("The list is empty");
        }
        else{
            while(current.data!=Key){
                current = current.next;
            }
            node.data = data;
            node.next = current.next;
            current.next = node;
        }
    }

    public void deleteAtStart(){
        Node temp;
         Node current;

        if(head==null){
            System.out.print("List is empty");
        }
        else{
            temp = head;
            current = temp.next;
            head.next=current;
        }
    }

    public void insertAtEnd(int data){
```

```
        Node node = new Node(data);
        Node current= head;

        if(head== null){
            System.out.println("The list is empty.");
        }
        else{
            while(current.next!=null){
                current = current.next;
            }
            node.data = data;
            current.next = node;
            node.next = null;

        }
    }

    public void deleteAtEnd(){
        Node current = head;
        Node temp = null;

        if(head==null){
            System.out.println("List empty");
        }
        else{
            while(current.next != null){
                temp = current;
                current = current.next;
            }
            temp.next = null;
        }
    }

    public void deleteDataKey(){
        Node current = head;
        int Key = 34;
        Node temp = null;
        if(head==null){
            System.out.println("Cant delete list is empty");
        }
        else{
            while(current.data!=Key){
                temp = current;
                current = current.next;
            }
```

```
            temp.next = current.next;
        }
    }

    public void deleteAfter(int key){

        if (head == null){
            System.out.printf("The list is empty");
        }

        Node temp = head;

        while(temp.next != null){
            if(temp.data == key ){
                temp.next = temp.next.next;
            }
            temp = temp.next;
        }
    }

    public static void main(String [] args){

        linkedlist Lists = new linkedlist();

        Lists.addNode(23);
        Lists.addNode(99);
        Lists.addNode(45);
        Lists.addNode(44);
        Lists.addNode(22);

         Lists.show();

        Lists.insertAtStart(12);

        Lists.show();

        Lists.InsertAfter(34);

        Lists.deleteAtStart();

        Lists.show();

        Lists.insertAtEnd(78);

        Lists.show();
```

```
        Lists.deleteAtEnd();

        Lists.show();

        Lists.deleteDataKey();

        Lists.deleteAfter(45);

        Lists.show();

        Lists.deleteBefore(45);

        Lists.show();
    }
}
```

Merging of two SLLLs

```
class LinkedList{

    public class Node{
        int data;
        Node next;

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

    public Node head = null;
    public Node tail = null;

    public void addNode(int data){
        Node node = new Node(data);

        if(head==null){
            head = node;
            tail = node;
        }
        else{
            tail.next = node;
            tail = node;
        }
```

```
    }

    public void merge(LinkedList list1, LinkedList list2){
        Node node = list1.head;

        while(node.next!=null){
            System.out.println(node.data);
            node = node.next;
        }
        if(node.next == null){
            node.next = list2.head;
            while(node.next!=null){
                System.out.println(node.data);
                node = node.next;
            }
        }
    }

    public static void main(String[] args){

        LinkedList list1 = new LinkedList();
        LinkedList list2 = new LinkedList();

        list1.addNode(01);
        list1.addNode(45);
        list1.addNode(99);
        list1.addNode(89);

        list2.addNode(70);
        list2.addNode(99);
        list2.addNode(56);
        list2.addNode(20);

        LinkedList mergedList = new LinkedList();
        mergedList.merge(list1, list2);
    }
}
```

2a)

INSERTING BEFORE DATA ELEMENT KEY.

1. Create a new node
2. Traverse to data KEY

3. Assign data to the new node
4. New node should point where previous node was pointing.
5. Previous node should point to new node.

```
METHOD
{
New = getnew();
Ptr = head
Prev_node = node

While( Ptr-> data != KEY){
Prev_node = Ptr;
Ptr = Ptr->LINK;
}

New-> LINK = Prev_node -> LINK || New node points where previous was pointing
Prev_node-> LINK = New  || previous node points where new node points

}
```

b)

ALGORITHM TO DELETE DATA ELEMENT KEY

Traverse to the data element KEY , what points to KEY must point to what KEY points to.

```
Method
Ptr1 = HEADER                    //start from HEADER node
Ptr = ptr1 -> LINK

While(ptr != NULL){              //traverse to the end
      If (ptr -> DATA != KEY){        //if the data is not found , move next
       Ptr1 = ptr                     //Stores the previous pointer
         Ptr = ptr -> LINK            //change pointer to the next node
}
Else
ptr -> LINK = ptr -> LINK        //link of the predecessor to point to the successor
}
```

ALGORITHM TO DELETE BEFORE ELEMENT KEY.

1. Create three variables.
2. One stores current node, second stores previous node and third stores second previous node.
3. Traverse through the nodes until element KEY.
4. Assign link of previous node to second previous node.
5. Make previous node point to null.

```
METHOD
{
current=node;
prev_node=null;
temp=null;
while ( current->data!=KEY)       ||Traverse until you reach element KEY
{
        temp=prev_node          ||Stores second previous
        prev_node=current       ||Stores previous node
        current=current->LINK   ||Change pointer to point to the next node
}
        temp->LINK= prev_node->LINK
        prev_node->LINK=null
}
```