SIKSHA 'O' ANUSANDHAN
DEEMED TO BE UNIVERSITY

Admission Batch: **Session:**

Laboratory Record

Computer Science Workshop 2 (CSE 3141)

Submitted by

Name: Saswat Mohanty

Registration No.: 1941012407

Branch: Computer Science and Engineering (CSE)

Semester: 4th Section: D

Department of Computer Science & Engineering

Faculty of Engineering & Technology (ITER)

Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030

INDEX

Assignment on Linked List

Q1) Write a program to create a single linked list and perform ~~insert~~ insert at first, insert at last, insert at any position, delete first, delete last, delete from any position and display operation on it.

Program:-

[[Node.java]] →

```
public class Node {
    int data;
    Node next;
    Node() {}
    Node(int data) {
        this.data = data;
    }
    Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
    public static void display(Node head) {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}
```

[EA491.java] →

```
import java.util.*;
public class A491{
   static Scanner sc = new Scanner(System.in);
   public static void create(Node head){
      Node p,q;
      p=head;
      System.out.println("Enter the value");
      p.data = sc.nextInt();
      p.next = null;
      System.out.println("Do you want to add another
                          node [yes-1, no-0]");
      int option = sc.nextInt();
      while(option != 0){
         q=new Node();
         System.out.println("Enter the value");
         q.data = sc.nextInt();
         q.next = null;
         p.next = q;
         p=q;
         System.out.println("Do you want to add another
                             node [yes-1, no-0]");
         option = sc.nextInt();
      }
      System.out.println("Linked list is created successfully");
}
```

```
public static Node InsBeg(Node head){
    Node p;
    p=new Node();
    System.out.println("Enter the value");
    p.data=sc.nextInt();
    p.next=null;
    if(head==null){
        head=p;
    }
    else{
        p.next=head;
        head=p;
    }
    return head;
}
public static Node InsEnd(Node head){
    Node p,q;
    p=new Node();
    System.out.println("Enter the value");
    p.data=sc.nextInt();
    p.next=null;
    if(head==null)
        head=p;
    else{
        q=head;
        while(q.next!=null)
            q=q.next;
        q.next=p;
    }
```

```
    return head;
}

public static Node insAfter (Node head) {
    Node p, q;
    System.out.println ("Enter the value after which you want to
                                    insert");
    int n = sc.nextInt();
    int flag = 0;
    q = head;
    while (q != null) {
        if (q.data = n) {
            flag = 1;
            break;
        }
        q = q.next;
    }
    if (flag == 1) {
        p = new Node();
        System.out.println("Enter the name of the new value");
        p.data = sc.nextInt();
        p.next = q.next;
        q.next = p;
    }
    else
        system.out.println ("Node is not found");
    return head;
}
```

```
public static Node DelBeg(Node head){
    Node p;
    if(head==null){
        System.out.println("Linked List is empty");
        System.exit(0);
    }
    else{
        p=head;
        head=head.next;
        p.next=null;
    }
    return head;
}
public static Node DelEnd(Node head){
    Node p,q;
    q=null;
    if(head==null){
        System.out.println("Linked list is empty");
        System.exit(0)
    }
    else{
        p=head;
        while(p.next!=null){
            q=p;
            p=p.next;
        }
        q.next=null;
    }
    return head;
}
```

```
public static Node delSpecific (Node start) {
    Node p,q;
    q = null;
    if(start == null) {
        System.out.println("The linked list is empty");
        System.exit(0);
    }
    else {
        System.out.println("Enter the value to be deleted:");
        int n = sc.nextInt();
        p = start;
        while (p != null) {
            if (p.data == n)
                break;
            q = p;
            p = p.next;
        }
        q.next = p.next
        p.next = null;
    }
    return start;
}

public static void display (Node head) {
    Node temp = head;
    while(temp != null) {
        System.out.println(temp.data+" ");
        temp = temp.next;
    }
```

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

public static void main (String [] args) {
    Node head = new Node ();
    while (true) {
        System.out.println ("***** MENU *****");
        System.out.println ("0: Exit");
        System.out.println ("1: Creation");
        System.out.println ("2: Insert a node at beginning");
        System.out.println ("3: Insert a node at end");
        System.out.println ("4: Insert after a given node");
        System.out.println ("5: Delete a node from beginning");
        System.out.println ("6: Delete a node from end");
        System.out.println ("7: Delete a specific node");
        System.out.println ("*********************");
        System.out.println ("Enter the choice: ");
        int choice = sc.nextInt();
        switch (choice) {
            case 0:
                System.exit(0);
            case 1:
                create (head);
                break;
            case 2:
                head = InsBeg (head);
                break
            case 3;
                head = InsEnd (head);
                break;
```

```
                case 4 :
                    head = InsAfter (head) ;
                    break;
                case 5 :
                    head = DelBeg(head);
                    break;
                case 6 :
                    head = DelEnd (head) ;
                    break;
                case 7 :
                    head = DelSpecify (head);
                    break;
                case 8:
                    display(head);
                    break;
                default :
                    System.out.println("Wrong choice");
            }
        }
    }
}
```

Output:-

```
***.****.**.* MENU *********
0: Exit
1: creation
2: Insert a node at beginning
3: Insert a node at end
4: Insert after a given node
5: Delete a node from beginning
6: Delete a node from end
```

7: Delete a specific node

8: Display

****

Enter the choice:

0

Q3) Write a program that takes two lists, assumed to be sorted, and returns their merge. The only field your program can change in a node is its next field.

Program :-

[[Node.java]] →

```
public class Node {
    int data;
    Node next;
    Node() {}
    Node (int data) {
        this.data = data;
    }
    Node (int data, Node next) {
        this.data = data;
        this.next = next;
    }
    public static void display (Node head) {
        Node temp = head;
        while (temp != null) {
            System.out.print (temp.data + " ");
            temp = temp.next;
        }
        System.out.println ();
    }
}
```

[A4Q2.java] →

```
public class A4Q2{
  Node head = null, tail = null;
  public void add(int data){
     Node newnode = new Node(data);
     if(head == null){
        head = newnode;
        tail = head;
        return;
     }
     tail.next = newnode;
     tail = newnode;
  }
  public Node mergeTwoLists (Node l1, Node l2) {
     if(l1 == null)
        return l2;
     if(l2== null)
        return l1;
     if (l1.data < l2.data){
        l1.next = mergeTwoLists (l1.next, l2);
        reloarn l1;
     }
     else {
        l2.next = mergeTwoLists (l1, l2.next);
        return l2;
     }
  }
  public static void main (String[] args) {
     A4Q2 ml = new A4Q2();
     A4Q2 ml2 = new A4Q2();
```

```
        ml.add(0);
        ml.add(1);
        ml.add(2);
        Node.display(ml.head);
        ml2.add(4);
        ml2.add(5);
        ml2.add(7);
        Node.display(ml2.head);
        ml.head = new A4Q2().mergeTwoLists(ml.head, ml2.head);
        Node.display(ml.head);
    }
}
```

Output:-

```
0 1 2
4 5 7
0 1 2 4 5 7
```

Q3) Write a program which takes a singly linked list L and two integers s and f as arguments, and reverse the order of the nodes from the s^{th} node ... to f^{th} node, inclusive. The numbering begins at 1 i.e., the head node is the first node. Do not allocate additional nodes.

Program :-

[[Node.java]] →

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

```
    Node (int data, Node next){
        this.data = data;
        this.next = next;
    }
    public static void display (Node head){
        Node temp = head;
        while(temp != null){
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}
```

[[A4Q3.java]] →

```
public class A4Q3{
  public static Node reverse(Node head, int s, int f){
    Node p = null;
    Node q = head;
    for (int i = 1; q != null && i < s; i++){
        p = q;
        q = q.next;
    }
    Node start = q;
    Node end = null;
    for (int i = 1; q != null && i <= f-s+1; i++){
        Node next = q.next;
        q.next = end;
        end = q;
        q = next;
    }
```

```
    start.next = q;
    if (p != null)
        p.next = end;
    else
        head = end;
    return head;
}
public static void main(String[] args) {
    int s = 2, f = 6;
    Node head = null;
    for (int i = 7; i >= 1; i--) {
        head = new Node(i, head);
    }
    Node.display(head);
    head = reverse(head, s, f);
    Node.display(head);
}
}
```

Output:-

```
1 2 3 4 5 6 7
1 6 5 4 3 2 7
```

Q4) Write a program that takes the head of a singly linked list and returns null if there does not exist a cycle, and the node at the start of the cycle, if a cycle is present. (You do not know the length of the list in advance.)

Program:-

[Node.java] →

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

```
    Node next;
    Node() {}
    Node(int data) {
        this.data = data;
    }
    Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
    public static void display(Node head) {
        Node temp = head;
        while(temp != null) {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }
}
```

[A4Q4.java] →

```
public class A4Q4 {
    static Node head;
    public static Node cycle() {
        Node slow = head, fast = head;
        boolean flag = false;
        while(slow != null && fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if(slow == fast) {
                flag = true;
                break;
```

```
        }
    }
    if (flag)
        return head;
    return null;
}

public static void main (String[] args) {
    head = null;
    for (int i = 3; i >= 1; i--) {
        head = new Node (i, head);
    }
    head.next.next.next = head;
    System.out.println (cycle());
}
}
```

Output:-

```
Node@2833cc44
```

Q5) Given a singly linked list and an integer K, write a program to remove the K^{th} last element from the list. Your algorithm cannot use more than a few words of storage, regardless of the length of the list. In particular, you can not assume that it is possible to record the length of the list.

Program:-

[Node.java] →

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

```
Node(int data, Node next){
    this.data = data;
    this.next = next;
}
public static void display(Node head){
    Node temp = head;
    while(temp != null){
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
    System.out.println();
}
}
```

[A405.java] →

```
public class A405{
  public static Node removeKth(Node head, int n){
    int size = 0;
    Node currNode = head;
    while(currNode != null){
        size++;
        currNode = currNode.next
    }
    size -= n;
    if(size == 0){
        head = head.next;
        return head;
    }
    Node temp = new Node();
    temp.next = head;
```

```
    while (size >= 1) {
        size--;
        temp = temp.next;
    }
    temp.next = temp.next.next;
    return head;
}
public static void main (String[] args) {
    Node head = null;
    for(int i=7; i>=1; i--) {
        head = new Node (i, head);
    }
    Node.display (head);
    removeKth (head, 3);
    Node.display (head);
}
}
```

Output :-

```
1 2 3 4 5 6 7
1 2 3 4 6 7
```

Q6) Write a program that takes as input a singly linked list of integers in sorted order, and removes duplicates from it. The list should be sorted.

Program :-

[[Node.java]] →

```
public class Node {
    int data;
    Node next;
    Node() {}
```

```
Node (int data) {
    this.data = data;
}
Node (int data, Node next) {
    this.data = data;
    this.next = next;
}
public static void display (Node head) {
    Node temp = head;
    while (temp != null) {
        System.out.print (temp.data + " ");
        temp = temp.next;
    }
    System.out.println ();
}
}
```

[A4Q6.java] →

```
public class A4Q6 {
  public static Node deleteduplicate (Node head) {
    Node temp = head;
    while (temp != null && temp.next != null) {
      if (temp.next.data == temp.data)
        temp.next = temp.next.next;
      else
        temp = temp.next;
    }
    return head;
  }
  public static void main (String [] args) {
```

```
Node head = null;
head = new Node (1, head);
head = new Node (2, head);
head = new Node (10, head);
head = new Node (10, head);
head = new Node (13, head);
head = new Node (13, head);
Node.display (head);
deleteduplicate (head);
Node.display (head);
}
}
```

Output :-

```
13  13  10  10  2  1
13  10  2  1
```

Q7) Write a program that tests whether a singly linked list is palindrome.

Program :-

[[Node.java]] →

```
public class Node {
    int data;
    Node next;
    Node() {}
    Node (int data) {
        this.data = data;
    }
    Node (int data, Node next) {
        this.data = data;
        this.next = next;
```

```
    }
    public static void display(Node head){
        Node temp = head;
        while(temp != null){
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }
}
```

[A4Q7.java] →

```
public class A4Q7{
    public static boolean isPalindrome(Node head){
        if(head == null)
            return true;
        Node rnode = reversedList(head);
        Node temp = ~~rnode~~ head;
        Node rtemp = rnode;
        while(rtemp != null){
            if(temp.data != rtemp.data)
                return false;
            temp = temp.next;
            rtemp = rtemp.next;
        }
        return true;
    }
    public static Node reversedList(Node head){
        Node rnode = new Node(head.data);
```

```
Node revnode = head.next;
rnode.next = null;
while (revnode != null) {
    Node temp = new Node (revnode.data);
    revnode = revnode.next;
    temp.next = rnode;
    rnode = temp;
}
return node;
}
public static void main (String[] args) {
    Node head = null;
    head = new Node (1, head);
    head = new Node (2, head);
    head = new Node (3, head);
    head = new Node (3, head);
    head = new Node (2, head);
    head = new Node (1, head);
    Node.display (head)
    if (isPallindrome (head))
        System.out.println ("Pallindrome");
    else
        System.out.println ("Not Pallindrome");
}
}
```

Output:-

1 2 3 3 2 1

Pallindrome.

08) Implement a function which takes as input a singly linked list and an integer k and performs a pivot of the list with respect to k. the relative ordering of nodes that appear before k, and after k, must remain unchanged; the same must hold for nodes holding keys equal to k.

Program :-

[Node.java] →

```
public class Node {
  int data;
  Node next;
  Node() {}
  Node (int data) {
     this.data = data;
  }
  Node (int data, Node next) {
     this.data = data;
     this.next = next;
  }
  public static void display (Node head) {
     Node temp = head;
     while (temp != null) {
        system.out.print(temp.data + " ");
        temp = temp.next;
     }
     System.out.println();
  }
}
```

[A498.java] →

```
public class A498 {
  public static Node listPivoting(Node head, int k) {
    Node l = new Node(0, null);
    Node e = new Node(0, null);
    Node g = new Node(0, null);
    Node l1 = l;
    Node e1 = e;
    Node g1 = g;
    Node newnode = head;
    while (newnode != null) {
      if (newnode.data < k) {
        l1.next = newnode;
        l1 = newnode;
      }
      else if (newnode.data == k) {
        e1.next = newnode;
        e1 = newnode;
      }
      else {
        g1.next = newnode;
        g1 = newnode;
      }
      newnode = newnode.next;
    }
    g1.next = null;
    e1.next = g.next;
    l1.next = e.next;
```

```
        return l.next;
    }
    public static void main(String[] args){
        Node head = null;
        head = new Node(1, head);
        head=new Node(4, head);
        head= new Node(5, head);
        head = new Node(3, head);
        head = new Node(2, head);
        head = new Node(5, head);
        Node.display(head);
        head= testPivoting(head, 3);
        Node.display(head);
    }
}
```

Output :-

5 2 3 5 4 1
2 1 3 5 5 4