IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student: N-Ravi Chandpita

Class: CSD-B Semester: IIIrd Semester

Course Code: ACSC11 Course Name: DS Laboratory

Roll Number									
2	3	9	5	1	A	6	7	8	3

Name of the Course Faculty: Mr. A. Srikanth Faculty ID: IARE10652

Exercise Number: 8 Week Number: 8 Date: 19-11-2024

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure Performance in the Lab		Source Code Calculations and Graphs	Program Execution Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	8-1	Circular linked list							
2	8-2	Doubly linked list							
3	8-3	Sorted merge of two Sorted doubly	4	2	2	4	4	4	20
4	8-4	Delete all occurrences of a given key							
5	8-5	Delete a doubly linked list node							
6									
7									
8									
9									
10									
11									
12									

N-Ravi Chandpita
Signature of the Student

Signature of the Faculty

8.1 Aim: The circular linked list is a linked list where all the nodes are connected to form a circle. In a circular linked list, the firstnode & lastnode are connected to eachother which forms a circle. There is no NULL at the end.

Code:

```
class Node {
      int data;
    Node next;
    Node (int data) {
          this.data = data;
          this.next = null;
    }
}
Class CircularLinkedList {
   private Node head = null;
   private Node tail = null;
   public void insertAtBeginning (int data) {
       Node newNode = new Node(data);
       if (head == null) {
           head = newNode;
           tail = newNode;
           tail.next = head;
       } else {
           newNode.next = head;
           tail.next = newNode;
           head = newNode;
       }
   }
```

```
public void insertAtEnd(int data) {
    Node newNode = new Node (data);
    if(head == null) {
        head = newNode;
        tail = newNode;
        tail.next = head;
    }else {
        tail.next = newNode;
        newNode.next = head;
        tail = newNode;
    }
}
public void insertAfter (int afterData, int data) {
    Node current = head;
    do {
        if (current.data == afterData) {
            Node newNode = new Node (data);
            newNode.next = current.next;
            current.next = newNode;
            if (current == tail) {
                tail = newNode;
            }
            return;
        }
        current = current.next;
    } while (current != head);
```

```
        System.out.println("Node with data" + afterdata + "not found.");
    }
    public void deleteAtBeginning() {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }
        if (head == tail) {
            head = null;
            tail = null;
        } else {
            head = head.next;
            tail.next = head;
        }
    }
public void deleteAtEnd() {
    if (head == null) {
        System.out.println("List is empty.");
        return;
    }
    if (head == tail) {
        head = null;
        tail = null;
    } else {
        Node current = head;
        while (current.next != tail) {
            current = current.next;
```

```
      }
      current.next = head;
      tail = current;
    }
  }
  public void deleteAfter(int afterdata) {
     Node current = head;
     do {
        if (current.data == afterData){
           if(current.next == head) {
              System.out.pointln("No node to delete " + afterData);
              return;
           }
           current.next = current.next.next;
           if(current.next == head) {
              tail = current;
           }
           return;
        }
        current = current.next;
     }while (current != head);
     System.out.pointln("Node with data" + afterData + "not found.");
  }
  public void traverse(){
     if (head == null) {
        System.out.pointln("List is empty.");
        return;
     }
```

```
        Node current = head;
        do {
            System.out.print(current.data + " ");
            current = current.next;
        } while (current != head);
        System.out.println();
    }
}
public class Circular_Linked_List {
    public Static void main(String[] args) {
        CircularLinkedList Cl1 = new CircularLinkedList();
        Cl1.insertAtBeginning(10);
        Cl1.insertAtBeginning(5);
        Cl1.insertAtEnd(20);
        cl1.insertAtEnd (25);
        Cl1.insertAfter(10,15);
        System.out.println("Circular List after insertions :");
        cl1.traverse();
        Cl1.deleteAtBeginning();
        Cl1.deleteAtEnd();
        Cl1.deleteAfter(10);
        System.out.println("Circular List after deletions :");
        cl1.traverse();
    }
}.
```

Output:

```
Circular linked list after insertions : 5 10 15 20 25
Circular linked list after deletions : 10 20.
```

Aim: A doubly linked list is a type of linked list in which each node consists of prev, data and next.

Code:

```
class Node {
    int data;
    Node prev;
    Node next;
    Node (int data) {
        this.data = data;
        this. prev = null;
        this. next = null;
    }
}
class DoublyLinkedList {
    private Node head = null;
    private Node tail = null;
    public void insertAtBeginning (int data) {
        Node newNode = new Node(data);
        if(head = null) {
            head = tail = newNode;
        } else {
```

```
        newNode.next = head;
        head.prev = newNode;
        head = newNode;
     }
  }
  public void insertAtEnd (int data) {
     Node newNode = new Node(data);
     if (tail == null) {
        head = tail = newNode;
     } else {
        tail.next = newNode;
        newNode.prev = tail;
        tail = new Node;
     }
  }
  public void insertAfter(int afterData, int data) {
     Node current = head;
     while (current != head && current.data != afterData) {
        current = current.next;
     }
     if (current == null) {
        System.out.println("Node with data");
        return;
     }
     Node newNode = new Node (data);
     newNode.next = current.next;
     newNode.prev = current;
     if (current.next != null) {
        current.next.prev = newNode;
    } else {
        tail = newNode;
    }
    current.next = newNode;
  }
  public void deleteAtBeginning() {
      if (head == null) {
          System.out.println("List is Empty.");
          return;
      }
      if(head == tail) {
          head = tail = null;
      } else {
          head = head.next;
          head.prev = null;
      }
  }
  public void deleteAtEnd() {
      if(tail == null) {
          System.out.println("List is Empty.");
          return;
      }
      if(head == tail) {
          head = tail = null;
      } else {
          tail = tail.prev;
          tail.next = null;
      }
  }
```

```
public void deleteNode (int data) {
    Node current = head;
    while (current != null && current.data != data) {
        current = current.next;
    }
    if (current == null) {
    System.out.println("Node with data" + data+"not fo
    return;
   }
   if (current = = head) {
       deleteAtBeginning();
  } else if (current == tail) {
      deleteAtEnd();
  } else {
      current.prev.next = current.next;
      current.next.prev = current.prev;
  }
}
public void traverseForward() {
    if (head == null) {
        System.out.println("List is Empty.");
        return;
    }
    Node current = head;
    While (current ! =null) {
        System.out.print(current.data+" ");
        current = current.next;
    }
    System.out.println();
}
```

```
  public void traverseBackward() {
    if (tail == null) {
      System.out.println(" List is Empty.");
      return;

    }
    Node current = tail;
    while (current != null) {
       System.out.print (current.data+" ");
       current = current.prev;
    }
    System.out.println();
  }

}
public class Doubly_Linked_List {
   public static void main (String[] args) {
      DoublyLinkedList dl1 = new DoublyLinkedList();
      dl1.insertAtBeginning (10 ; 15);
      dl1.insertAtEnd (20.25);
      dl1.insertAfter (10,15).
      dl1.TraverseForward();
      dl1.TraverseBackward();
      dl1.deleteNode(15);
    System.out.printlnf("Doubly List (Forward Traversal):");
    System.out.println(" Linked List -After deletions (Forward
                                        Traversal):");

   }

}
```

Output:

```
Doubly linked list (Forward Traversal): 5 10 15 20 25,
Doubly linked list (Backward Traversal): 25 20 15 10 5
Doubly linked list After deletion (Forward Traversal): 10
Doubly linked list After deletion (Backward Traversal): 20 10
```

8.3 Aim: Given two sorted Doubly circullar linked list tha containing n1 & n2 nodes respectively. the problem is t merge the two lists such that resultant list is also sorted order.

Code:

```
class Node {
    int data;
    Node next;
    Node prev;
    Node (int data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}
class DoublyCircularLinkedList {
    Node head;
    public void insertAtEnd (int data) {
        Node newNode = new Node (data);
        if(head == null) {
            head = newNode;
        head.next = head;
        head.prev = head;
    } else {
        Node last = head.prev;
        last.next = newNode;
        newNode.prev = last;
        newNode.next = head;
        head.prev = newNode;
    }
}
public void display() {
    if (head == null) {
        System.out.println("List is Empty.");
        return;
    }
    Node current = head;
    do {
        System.out.print(current.data+" ");
        current = current.next;
    } while (current != head);
    System.out.println();
}
public static Node mergeLists(Node head1, Node head2)
    if(head1 == null) return head2;
    if(head2 == null) return head1;
    Node last1 = head1.prev;
    Node last2 = head2.prev;
    Node lastNode = (last1.data < last2.data) ? last2 : last1;
    last1.next = null;
    last2.next = null;
    Node finalHead = mergesortedDoublyLists(head1, head2);
    finalHead.prev = lastNode;
    lastNode.next = finalHead;
    return finalHead;
}
private static Node mergesortedDoublyLists(Node head1, Node
                                                   head2) {

    Node dummy = new Node(0);
    Node tail = dummy;
    while(head1! = null && head2 ! = null) {
        if(head1.data < = head2. data) {
            tail.next = head1;
            head1.prev = tail;
            head1 = head1.next;
        } else {
            tail.next = head2;
            head2. prev = tail;
            head2 = head2.next;
        }
        tail = tail.next;
    }
    if(head1 != null) {
      tail.next = head1;
```

```
        head1.prev = tail;
      }
      if(head2 != null) {
        tail.next = head2;
        head2.prev = tail;
      }
      Node mergedHead = dummy.next;
      if (mergedHead != null){
         mergedHead.prev = null;
      }
      return mergedHead;
    }
}
public class Merge2sortedDoublyCL1 {
    public static void main (String[] args) {
       DoublyCircularLinkedList list1 = new DoublyCL1();
       DoublyCircularLinkedList list2 = new DoublyCL2();
       list1.insertAtEnd(1);
       list1.insertAtEnd(3);
       System.out.println("List 1: ");
       list1.display();
       # Similarly list 2
       Node mergedHead = DoublyCircularLinkedList.mergelists
                                        (list1.head, list2.head);
       System.out.println("Merged list: ");
       mergedList.display();
    }
}.
```

Output:

```
List 1 : 1 3 5
List 2 : 2 4 6
Merged List : 1 2 3 4 5 6.
```

8.5 Aim: Given a doubly linked list and a key x. The problem is to delete all occcuences of given key x from the doubly linked list.

Code:

```
class Node {
    int data;
    Node next, prev;
    Node (int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
public class DeleteNodeAtGivenPosition {
    Static Node deleteNode(Node head, Node del) {
        if (del == null) {
            return head;
        }
        if (head == del) {
            head = del.next;
        }
    if(del.prev != null) {
        del.prev.next = del.next;
    }
    if(del.next != null) {
        del.next.prev = del.prev;
    }
    del = null;
    return head;
}
static Node deleteNodeAtGivenPos(Node head, int n) {
    if (head == null || n<=0) {
        return head;
    }
    Node current = head;
    for(int i=1; i<n && current != null; i++) {
        current=current.next;
    }
    if (current == null) {
        System.out.println("Position "+n+" is out of range.");
        return head;
    }
    head = deleteNode(head, current);
    return head;
}
static void printList(Node head) {
    if (head == null) {
        System.out.println("List is empty.");
        return;
    }
    }
    Node current = head;
    while(current != null) {
        System.out.print(current.data + " ");
    }
    System.out.println();
  }
  public static void main(String[] args) {
    Node head = new Node(1);
    head.next = new Node(2);
    head.next.prev = head;
    head.next.next = new Node (3);
    head.next.next.prev = head.next;
    head.next.next.next = new Node(4);
    head.next.next.next.prev = head.next.next;
    head.next.next.next.next = new Node(5);
    head.next.next.next.next.prev = head.next.next.next;
    System.out.println("Original List:");
    printList(head);
  }
}.
```

Output:

```
Original List : 1 2 3 4 5
List After deleting node at position 2: 1 3 4 5
```

18|8|1

4 Aim: Given a doubly linked list and a position n. The task is to delete the node at the given position n from the beginning.

Code:

```
Class Node {
    int data;
    Node next, prev;
    Node (int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
public class DeleteOccurenceInDoublyLinkedList {
    Static Node deleteNode (Node head, Node delNode) {
        if (head == delNode) {
            head = delNode.next;
        }
        if (delNode.next != null) {
            delNode.next.prev = delNode.prev;
        }
        if (delNode.prev != null) {
            delNode.prev.next = delNode.next;
        }
        delNode = null;
        return head;
    }
```

```
static Node deleteAllOccurOfx(Node head, int x) {
    if (head == null) {
        return null;
    }
   Node current = head;
   while (current != null) {
       Node next = current.next;
       if (current.data == x) {
          head = deleteNode(head, current);
       }
       current = next;
   }
   return head;
}
static void printList(Node head) {
    if (head == null) {
        System.out.println("List is Empty.");
        return;
    }
    Node current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main (String args[]) {
```

```
Node head = new Node(2);
head.next = new Node(2);
head.next.prev = head;
head.next.next = new Node(10);
head.next.next.prev = head.next ;
head.next.next.next = new Node(8);
head.next.next.next.prev = head.next.next ;
head.next.next.next.next = new Node(4);
head.next.next.next.next.prev = head.next.next.next;
head.next.next.next.next.next = new Node(2);
head.next.next.next.next.next.next = head.next.next.next;
System.out.println("Original List:");
printList(head);
int x=2;
head = deleteAllOccurOfX(head, x);
System.out.println("List after deleting all occurences of"+x+":");
printList(head);
}
}.
```

Output:

Original List: 2 2 10 8 4 2 5 2

List after deleting all occurences of 2: 10 8 4 5.