IARE

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

LABORATORY WORK BOOK

Name of the Student : Racherla Santhosh

Class : IT-B Semester : 03

Course Code : ACSD11 Course Name : DS laboratory

Name of the Course Faculty : Ms. K. Laxminarayanamma Faculty ID : IARE10033

Exercise Number : 08 Week Number : 08 Date : 04/11/2024

Roll Number									
2	3	9	5	1	A	1	2	C	3

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	U	U	-	U	U	U	20
2	8.2	Doubly Linked list							
3	8.3	Sorted Merge of Two Sorted DLL							
4	8.4	Delete All occurences of A given Key in DLL							
5	8.5	Delete A DLL Node At A Given position.							
6									
7									
8									
9									
10									
11									
12									

Santhosh

Signature of the Student

Signature of the Faculty

8. Circular Single Linked List and Doubly Linked List.

8.1 Circular Linked List :-

AIM :- Write a Program On Circular Linked List for doing Operations on list.

PROGRAM :-

```
import java.util.Scanner;
class CircularLinkedList {
  Static class Node {
    int data;
    Node next;
    Public Node (int data) {
      this.data = data;
      this.next = null;
    }
}
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;
        head = newNode;
        tail.next = head;
    }
}
Public void insertAtEnd(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
        tail = newNode;
        tail.next = head;
    } else {
        tail.next = newNode;
        tail = newNode;
        tail.next = head;
    }
}
```

```
Public void insertAfter (int data, int position) {
    Node newNode = new Node (data);
    Node current = head;
    int count = 1;
    do {
        if (count == position) {
            newNode.next = current.next;
            current.next = newNode;
            if (current == tail) {
                tail = newNode;
            }
            return;
        }
        current = current.next;
        count++;
    } while (current != head);
    System.out.println ("Position out of range");
}

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 deleteAtPosition (int position ) {
  if (head == null) {
    System.out.println(" List is Empty ");
    return;
  }
  if ( Position == 1) {
    deleteAtBeginning();
    return;
  }
  Node current = head;
  Node previous = null;
  int count = 1;
  do {
    if (count == position) {
      previous.next = current.next;
      if ( current == tail) {
        tail = previous;
      }
      return;
    }
    previous = current;
    current = current.next;
    count ++;
```

```
    } while (current != head);
    System.out.println("Position out of range");
}
Public void traverse() {
    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 void main(String[] args) {
    CircularLinkedList cll = new CircularLinkedList();
    Scanner Scanner = new Scanner(System.in);
    int choice, data, Position;
    do {
        System.out.println("\n-- Circular Linked List OP --");
        System.out.println("1. Insert At Beginning");
```

```
System.out.println("2. Insert At End");
System.out.println("3. Insert After position");
System.out.println("4. Delete At Beginning");
System.out.println("5. Delete At End");
System.out.println("6. Delete At position");
System.out.println("7. Traverse");
System.out.println("8. Display List");
System.out.println("9. Exit");
System.out.println("Enter your choice : ");
choice = Scanner.nextInt();
Switch (choice) {
  case 1:
    System.out.print("Enter data to insert at begin :");
    data = Scanner.nextInt();
    cll.insertAtBeginning(data);
    break;
  case 2:
    System.out.print("Enter data to insert at end :");
    data = Scanner.nextInt();
```

```
cll.insertAtEnd(data);
break;
case 3:
System.out.print("Enter data to insert:");
data = Scanner.nextInt();
System.out.print("Enter Position after which to
                  insert:");
Position = Scanner.nextInt();
cll.insertAfter(data, Position);
break;
case 4:
System.out.println("Deleting node at beginning....");
cll.deleteAtBeginning();
break;
case 5:
System.out.println("Deleting node at end...");
cll.deleteAtEnd();
break;
case 6:
System.out.print("Enter Position to delete:");
Position = Scanner.nextInt();
```

```
            cll.deleteAtPosition(Position);
            break;
        case 7:
            System.out.println("Traversing the Circular
                                Linked List:");
            cll.traverse();
            break;
        case 8:
            System.out.println("Displaying the Circular
                                Linked List:");;
            cll.traverse();
            break;
        case 9:
            System.out.println("Exiting....");
            break;
        default:
            System.out.println("Invalid choice. please Try again");
            break;
        }
    } while (choice != 9);
    Scanner.close(); }
}
```

OUTPUT :-

--- Circular Linked List Operations ---

1. Insert at Beginning
2. Insert at End
3. Insert After position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Traverse
8. Display List
9. Exit

Enter your choice : 1

Enter data to insert at Beginning : 1

--- circular Linked List Operations ---

Enter your choice : 2

Enter data to Insert at End : 3

--- Circular Linked List Operations ---

Enter your choice : 3

Enter data to insert : 2

Enter position after which to insert : 1

--- Circular Linked list Operations ---

Enter your choice : 8

Displaying the Circular Linked list : 1 2 3

--- Circular Linked list Operations ---

Enter your choice : 4

Deleting node at beginning....

--- Circular Linked list Operations ---

Enter your choice : 5

Deleting node at end ....

--- Circular Linked list Operations ---

Enter your choice : 6

Enter position to delete : 1

--- Circular Linked list Operations ---

Enter your choice : 7

Traversing the Circular Linked List :

List is empty

--- Circular Linked list Operations ---

Enter your choice : 9

Exiting ...

8.2 # Doubly Linked List :-

AIM :- Write a Program On Doubly Linked List for doing Operations on the list.

PROGRAM :-

```
import java.util.Scanner;
class Doubly LinkedList {
    Static class Node {
        int data;
        Node prev;
        Node next;
        Public Node(int data) {
            this.data = data;
            this.prev = null;
            this.next = null;
        }
    }
    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 (head == null) {
        head = tail = newNode;
    } else {
        tail.next = newNode;
        newNode.prev = tail;
        tail = newNode;
    }
}

Public void insertAfter (int data, int position) {
    Node newNode = new Node (data);
    Node current = head;
```

```
int count = 1;
while ( current != null && count < position ) {
    current = current.next;
    count++;
}
if ( current == null ) {
    System.out.println ("Position out of range");
} else {
    newNode.next = current.next;
    newNode.prev = current;
    if ( current.next != null ) {
        current.next.prev = newNode;
    }
    current.next = newNode;
    if ( current == tail ) {
        tail = newNode;
    }
}
}
```

```
Public void deleteAtBeginning() {
    if (head == null) {
        System.out.println ("List is empty");
```

```
} else if (head == tail) {
    head = tail = null;
} else {
    head = head.next;
    head.prev = null;
}
}
Public void deleteAtEnd() {
    if (head == null) {
        System.out.println("List is Empty");
    } else if (head == tail) {
        head = tail = null;
    } else {
        tail = tail.prev;
        tail.next = null;
    }
}
Public void deleteAtPosition (int Position) {
    if (head == null) {
        System.out.println("List is empty");
    } return;
```

```
Node current = head;
int count = 1;
while (current != null && count < position) {
    current = current.next;
    count++;
}
if (current == null) {
    System.out.println("Position out of range");
} else if (current == head) {
    deleteAtBeginning();
} else if (current == tail) {
    deleteAtEnd();
} else {
    current.prev.next = current.next;
    if (current.next != null) {
        current.next.prev = current.prev;
    }
}
}
Public void traverse() {
    if (head == null) {
        System.out.println("List is empty");
```

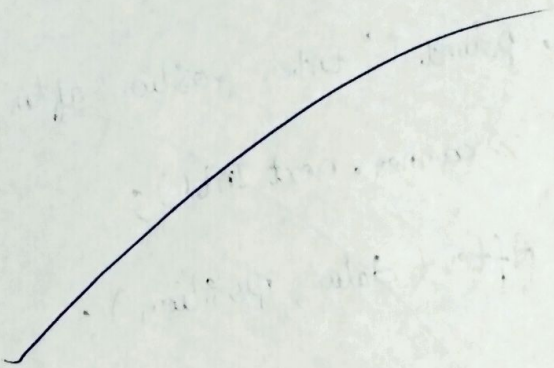
```
        return;
    }
    Scanner Scanner = new Scanner (System.in);
    System.out.print (" Enter 1 for Forward Traversal,
                        2 for Reverse Traversal : ");
    int direction = Scanner.nextInt();
    if (direction == 1) {
        System.out.print (" Forward Traversal : ");
        Node current = head;
        while (current != null) {
            System.out.print ( current.data + " ");
            current = current.next;
        }
        System.out.println();
    } else if ( direction == 2) {
        System.out.print (" Reverse Traversal : ");
        Node current = tail;
        while ( current != null) {
            System.out.print ( current.data + " ");
```

```
        current = current.prev;
    }
    System.out.println();
} else {
    System.out.println("Invalid choice");
}
}

Public void display() {
    if (head == null) {
        System.out.println("List is empty");
        return;
    }
    Node current = head;
    System.out.print("Doubly Linked List: ");
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}

Public static void main(String[] args) {
    Doubly Linked List dll = new Doubly Linked List();
```

```
Scanner Scanner = new Scanner (System.in);
int choice, data, position;
do {
  System.out.println ("\n--- Doubly Linked List Operations ---");
  System.out.println ("1. Insert at Beginning");
  System.out.println ("2. Insert at End");
  System.out.println ("3. Insert After Position");
  System.out.println ("4. Delete at Beginning");
  System.out.println ("5. Delete at End");
  System.out.println ("6. Delete at Position");
  System.out.println ("7. Traverse List");
  System.out.println ("8. Display List");
  System.out.println ("9. Exit");
System.out.print ("Enter Your choice: ");
choice = Scanner.nextInt();
Switch (choice) {
  case 1:
    System.out.print ("Enter data to insert at beginning: ");
```

```
data = Scanner.nextInt();
dll.insertAtBeginning(data);
break;
case 2:
System.out.print("Enter data to insert at end:");
data = Scanner.nextInt();
dll.insertAtEnd(data);
break;
case 3:
System.out.print("Enter data to insert:");
data = Scanner.nextInt();
System.out.print("Enter position after which to insert:");
position = Scanner.nextInt();
dll.insertAfter(data, position);
break;
case 4:
System.out.println("Deleting node at beginning...");
dll.deleteAtBeginning();
break;
```

```
case 5 :
    System.out.println (" Deleting node at end ... ");
    dll.deleteAtEnd();
    break;
case 6 :
    System.out.print (" Enter position to delete : ");
    position = Scanner.nextInt();
    dll.deleteAtPosition (position);
    break;
case 7 :
    dll.traverse();
    break;
case 8 :
    dll.display();
    break;
case 9 :
    System.out.println (" Exiting ... ");
    break;
default :
    System.out.println (" Invalid choice. Please try again");
```



```
            break;
        }
    } while (choice != 9);
    Scanner.close();
    }
}
```

OUTPUT :-

--- Doubly Linked List Operations ---

1. Insert at Beginning
2. Insert at End
3. Insert After Position
4. Delete at Beginning
5. Delete at End
6. Delete at Position
7. Traverse List
8. Display List
9. Exit

Enter your choice : 1

Enter data to insert at beginning : 1

--- Doubly Linked List Operations ---

```
Enter your choice : 2

Enter data to insert at end : 3

--- Doubly Linked List Operations ---

Enter your choice : 3

Enter data to insert : 2

Enter Position after which to insert : 1

--- Doubly Linked list Operations ---

Enter your choice : 8

Doubly Linked list : 1 2 3

--- Doubly Linked list Operations ---

Enter your choice : 4

Deleting node at beginning...

--- Doubly Linked list Operations ---

Enter your choice : 5

Deleting node at end...

--- Doubly Linked list Operations ---

Enter your choice : 6

Enter Position to delete : 1
```

Enter your choice : 7

List is empty

Enter your choice : 9

Exiting ...

8.3 Sorted Merge Of Two Sorted Doubly Circular Linked Lists :-

AIM :- Given two Sorted Doubly Circular Linked List Containing n1 and n2 nodes respectively. The problem is to merge the two lists such that resultant list is also in Sorted Order.

PROGRAM :-

```
import java.util.Scanner;
class DoublyCircularLinkedList {
    static class Node {
```

```
    int data;
    Node prev, next;
    Public Node (int data) {
        this.data = data;
        this.prev = this.next = this;
    }
}
Private Node head = null;
Public void insert (int data) {
    Node newNode = new Node (data);
    if (head == null) {
        head = newNode;
    } else {
        Node last = head.prev;
        newNode.next = head;
        head.prev = newNode;
        newNode.prev = last;
        last.next = newNode;
    }
}
```

```
Private Node getLastNode() {
    return head == null ? null : head.prev;
}
Public Static DoublyCircularLinkedList merge (DoublyCircularLinked
    List list1, DoublyCircularLinkedList list2) {
  if (list1.head == null) return list2;
  if (list2.head == null) return list1;
  Node last1 = list1.getLastNode();
  Node last2 = list2.getLastNode();
  last1.next = last2.next = null;
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 void main (String[] args) {
    Scanner Scanner = new Scanner (System.in);
Doubly Circular LinKed List list1 = new Doubly Circular LinKed List ();
Doubly Circular LinKed list list2 = new Doubly Circular LinKedlist();
System.out.println ("Enter elements for list 1 (enter -1 to
                    Stop) : ");
while (true) {
    int data = Scanner.nextInt();
    if (data == -1) break;
    list1.insert (data);
}
System.out.println ("Enter elements for list 2 (enter -1 to
                    Stop) : ");
while (true) {
    int data = Scanner.nextInt();
    if (data == -1) break;
```

```
        list2.insert(data);
    }
    System.out.println("List 1: ");
    list1.display();
    System.out.print("List 2: ");
    list2.display();
    DoublyCircularLinkedList mergedList = DoublyCircularLinkedList.
                                          merge(list1, list2);
    System.out.println("Merged List: ");
    mergedList.display();
    scanner.close();
  }
}
```

OUTPUT :-

```
Enter elements for List 1 (enter -1 to stop):
1 3 5 8 -1
Enter elements for List 2 (enter -1 to stop):
2 7 9 11 -1
```

List 1 : 1 3 5 8

List 2 : 2 7 9 11

Merged list : 1 2 3 5 7 8 9 11

8.4 Delete All Occurences Of A given Key in a Doubly Linked list :-

AIM :- Given a Doubly Linked List and a Key x. The problem is to delete all occurences of the given key x from the doubly linked list.

PROGRAM :-

```
import java.util.Scanner;
class DoublyLinkedList {
  Static class Node {
    int data;
    Node prev, next;
    Public Node ( int data ) {
      this.data = data;
```

```
        node.prev.next = node.next;
    }
    if (node.next != null) {
        node.next.prev = node.prev;
    }
}

Public Static void main (String[] args) {
    Scanner Scanner = new Scanner (System.in);
    Doubly LinkedList list = new Doubly LinkedList();
    System.out.println (" Enter elements of the doubly linked list
                        (enter -1 to Stop): ");
    While (true) {
        int data = Scanner.nextInt();
        if (data == -1) break;
        list.insert (data);
    }
    System.out.println (" Original List: ");
    list.display();
    System.out.print (" Enter the key to delete: ");
    int key = Scanner.nextInt();
```

```
    this.prev = null;
    this.next = null;
  }
}
Private Node head;
Public void deleteAllOccurrences(int key) {
  Node current = head;
  While (current != null) {
    if (current.data == key) {
      Node next = current.next;
      deleteNode(current);
      current = next;
    } else {
      current = current.next;
    }
  }
}
Private void deleteNode(Node node) {
  if (node == head) {
    head = node.next;
  }
  if (node.prev != null) {
```

PROGRAM :-

```
import java.util.Scanner;
class DoublyLinkedList {
  static class Node {
    int data;
    Node prev, next;
    public Node (int data) {
      this.data = data;
      this.prev = null;
      this.next = null;
    }
  }
  Private Node head;
  Public void DeleteAtPosition (int Position) {
    if (head == null || position <= 0) {
      System.out.println ("Invalid Position or empty list.");
      return;
    }
    Node current = head;
```

```
        list.deleteAllOccurrences(key);
        System.out.println("List after deleting occurrences of "+key+":");
        list.display();
        scanner.close();
    }
}
```

OUTPUT :-

Enter elements of the doubly linked list (enter -1 to stop):

2 2 10 8 4 2 5 2 -1

Original List: 2 <-> 2 <-> 10 <-> 8 <-> 4 <-> 2 <-> 5 <-> 2

Enter the key to delete: 2

List after deleting occurrences of 2: 10 <-> 8 <-> 4 <-> 5.

8.5 Delete A Doubly Linked List Node At A Given Position :-

AIM :- Given A Doubly linked list and a Position n. The task is to delete the node at the given Position n from beginning.

PROGRAM :-

```
import java.util.Scanner;
class DoublyLinkedList {
  Static class Node {
    int data;
    Node prev, next;
    Public Node (int data) {
      this.data = data;
      this.prev = null;
      this.next = null;
    }
}
Private Node head;
Public void DeleteAtPosition (int Position) {
  if (head == null || Position <= 0) {
    System.out.println ("Invalid Position or empty list.");
    return;
}
Node current = head;
```

```
for (int i = 1; current != null && i < position; i++) {
    current = current.next;
}
if (current == null) {
    System.out.println("Position exceeds the Size of list");
    return;
}
if (current == head) {
    head = current.next;
    if (head != null) {
        head.prev = null;
    }
} else {
    if (current.next != null) {
        current.next.prev = current.prev;
    }
    if (current.prev != null) {
        current.prev.next = current.next;
    }
}
Public Static void main(String[] args) {
```

```
Scanner scanner = new Scanner (System.in);
Doubly LinkedList list = new Doubly LinkedList();
System.out.println("Enter elements of the doubly linked
                    list (enter -1 to stop):");
While (true) {
    int data = scanner.nextInt();
    if (data == -1) break;
    list.insert(data);
}
System.out.println("Original List:");
list.display();
System.out.print("Enter the position to delete
                  (1-based index):");
int position = scanner.nextInt();
list.deleteAtPosition(Position);
System.out.println("List after deletion at position"+
                   Position + ":");
list.display();
```

```
        Scanner.close();
    }
}
```

OUTPUT :-

```
Enter elements of the doubly linked list (enter -1 to stop):
10 8 4 2 5 -1.
Original List : 10 <-> 8 <-> 4 <-> 2 <-> 5
Enter the position to delete (1-based index) : 2
List after deletion at position 2 : 10 <-> 4 <-> 2 <-> 5.
```

VIVA VOCE :-

1) What is Circular Linked List ?

A) The Circular Linked List is a linked list where all nodes are connected to form a circle. In a Circular Linked List, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.

2) What is Doubly Linked List ?

A) A Doubly Linked List is a type of linked list

in which each node consists of 3 components :

(a) * prev - address of the previous node

(b) data - data item

(c) * next - address of next node.

3) What is Sorted List ?

A) A Sorted List is a list where all elements are arranged in a specific order, usually ascending (or) descending. Each new element added is placed in the correct order to maintain this sequence.

4) What is Node Deletion ?

A) Node Deletion is the process of removing a specific node from a data structure while adjusting pointers or connections to maintain the structure's integrity.