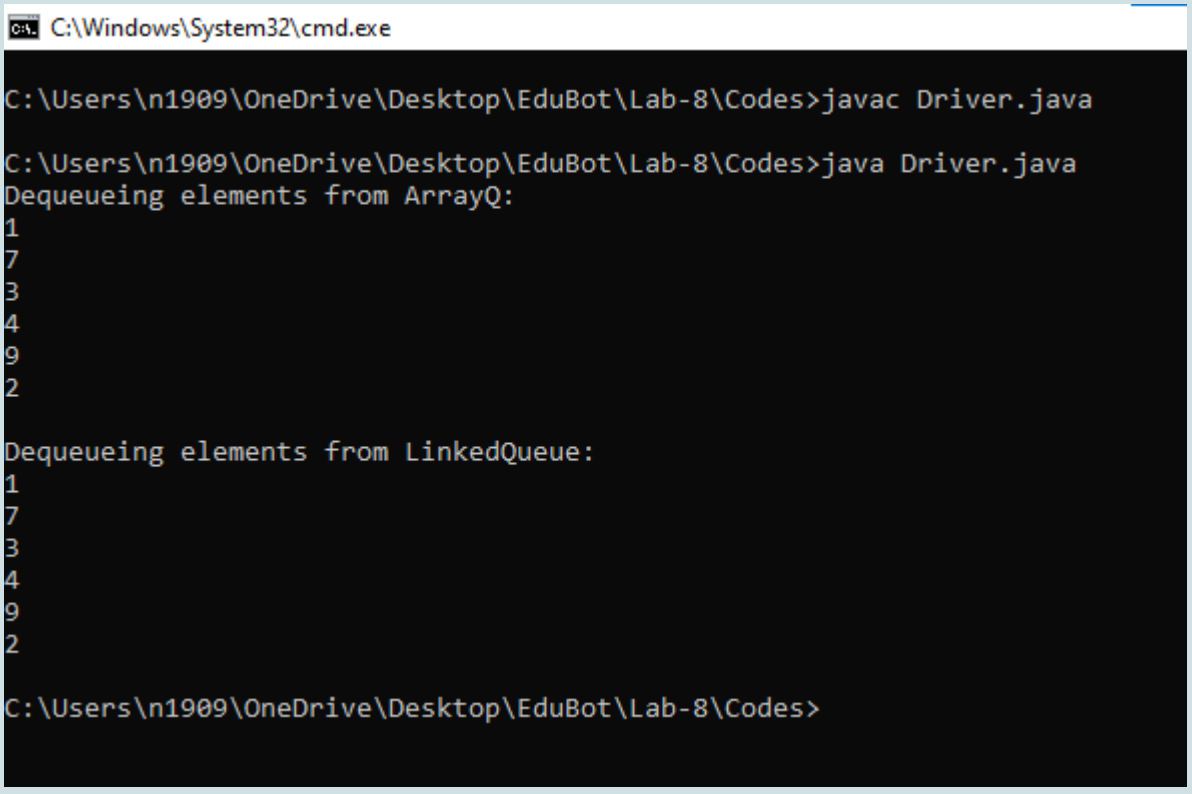
Programming Fundamentals
Lab #8

Exercise 1:

```
public class Ex1_Driver {
    public static void main(String[] args) {
        ArrayQ arrayQ = new ArrayQ();
        LinkedQueue linkedQueue = new LinkedQueue();

        // Enqueue elements
        int[] elements = {1, 7, 3, 4, 9, 2};
        for (int element : elements) {
            arrayQ.enqueue(element);
            linkedQueue.enqueue(element);
        }

        // Dequeue and display elements from ArrayQ
        System.out.println("Dequeueing elements from ArrayQ:");
        while (!arrayQ.isEmpty()) {
            System.out.println(arrayQ.dequeue());
        }

        // Dequeue and display elements from LinkedQueue
        System.out.println("\nDequeueing elements from LinkedQueue:");
        while (!linkedQueue.isEmpty()) {
            System.out.println(linkedQueue.dequeue());
        }
    }
}

class ArrayQ {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    public ArrayQ() {
        capacity = 10; // Default capacity
        queue = new int[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }
```

```
    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == capacity;
    }

    public void enqueue(int item) {
        if (isFull()) {
            System.out.println("Queue is full");
            return;
        }
        rear = (rear + 1) % capacity;
        queue[rear] = item;
        size++;
    }

    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return -1;
        }
        int item = queue[front];
        front = (front + 1) % capacity;
        size--;
        return item;
    }
}

class LinkedQueue {
    private Node front;
    private Node rear;

    private class Node {
        int data;
        Node next;

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

    public LinkedQueue() {
        front = null;
        rear = null;
    }
```

```
    public boolean isEmpty() {
        return front == null;
    }

    public void enqueue(int item) {
        Node newNode = new Node(item);
        if (isEmpty()) {
            front = newNode;
        } else {
            rear.next = newNode;
        }
        rear = newNode;
    }

    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return -1;
        }
        int item = front.data;
        front = front.next;
        if (front == null) {
            rear = null;
        }
        return item;
    }
}
```



Exercise 2:

```
public class Ex2_Driver {
    public static void main(String[] args) {
        ArrayQ arrayQ = new ArrayQ();
        LinkedQueue linkedQueue = new LinkedQueue();

        // Enqueue elements
        int[] elements = {1, 7, 3, 4, 9, 2};
        for (int element : elements) {
            arrayQ.enqueue(element);
            linkedQueue.enqueue(element);
        }

        // Remove middle element
        linkedQueue.removeMiddle();

        // Dequeue and display elements from ArrayQ
        System.out.println("Dequeueing elements from ArrayQ:");
        while (!arrayQ.isEmpty()) {
            System.out.println(arrayQ.dequeue());
        }

        // Dequeue and display elements from LinkedQueue
        System.out.println("\nDequeueing elements from LinkedQueue:");
        while (!linkedQueue.isEmpty()) {
            System.out.println(linkedQueue.dequeue());
        }
    }
}

class ArrayQ {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    public ArrayQ() {
        capacity = 10; // Default capacity
        queue = new int[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == capacity;
    }
```

```
    public void enqueue(int item) {
        if (isFull()) {
            System.out.println("Queue is full");
            return;
        }
        rear = (rear + 1) % capacity;
        queue[rear] = item;
        size++;
    }

    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return -1;
        }
        int item = queue[front];
        front = (front + 1) % capacity;
        size--;
        return item;
    }
}

class LinkedQueue {
    private Node front;
    private Node rear;
    private int size;

    private class Node {
        int data;
        Node next;

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

    public LinkedQueue() {
        front = null;
        rear = null;
        size = 0;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void enqueue(int item) {
        Node newNode = new Node(item);
        if (isEmpty()) {
            front = newNode;
        } else {
```

```
            rear.next = newNode;
        }
        rear = newNode;
        size++;
    }

    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return -1;
        }
        int item = front.data;
        front = front.next;
        if (front == null) {
            rear = null;
        }
        size--;
        return item;
    }

    public void removeMiddle() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return;
        }

        Node slowPtr = front;
        Node fastPtr = front;
        Node prevPtr = null;

        while (fastPtr != null && fastPtr.next != null) {
            fastPtr = fastPtr.next.next;
            prevPtr = slowPtr;
            slowPtr = slowPtr.next;
        }

        // If the size of the queue is even, slowPtr is pointing to the second
middle element
        // In this case, we want to remove the element before slowPtr
        if (prevPtr != null) {
            prevPtr.next = slowPtr.next;
        } else {
            // If the size of the queue is odd, slowPtr is pointing to the
middle element
            // We need to remove this element
            front = slowPtr.next;
        }

        if (slowPtr == rear) {
            rear = prevPtr;
        }

        size--;
```

```
    }
}
```

```
Select C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\n1909\OneDrive\Desktop\EduBot\Lab-8\Codes>javac Ex2_Driver.java

C:\Users\n1909\OneDrive\Desktop\EduBot\Lab-8\Codes>java Ex2_Driver.java
Dequeueing elements from ArrayQ:
1
7
3
4
9
2

Dequeueing elements from LinkedQueue:
1
7
3
9
2

C:\Users\n1909\OneDrive\Desktop\EduBot\Lab-8\Codes>
```

Exercise 3:

1. **What is the root node of the tree?**
    a. 50 is the Root Node
2. **What are the leaf nodes of the tree?**
    a. 9, 14, 19, 67, 76 are the the leaf Node for the given binary tree
3. **What are the ancestor nodes of the node containing 19?**
    a. 23, 17, 50 are the ancestor nodes for the node 19
4. **Write the sequence of node values that you would get from a post-order traversal.**
    a. 9, 14, 12, 19, 23, 17, 67, 54, 76, 72, 50 sequence of the node values post-order traversal