1. **Write a java program to demonstrate the working of Singly Linked List.**

CODE: -

```
package myStack;
//java program to implement singly linked list
import java.util.Scanner;

class Node4{
	int data;
	Node4 next;

	public Node4(int data) {
		this.data = data;
		this.next = null;
	}
}
public class SinglyLinkedList {
	// defining the head and tail of a singly linked list
		Node4 head;
		Node4 tail;

		public SinglyLinkedList() {
			head = null;
			tail = null;
		}

		public boolean isEmpty() {
			return (head == null);
		}

		// function to add a node to the list
		public void insert(int data) {
			Node4 newNode = new Node4(data);
			// if(isEmpty()){}
			if (head == null) {
				head = newNode;
				tail = newNode;
			} else {
				tail.next = newNode;
				tail = newNode;
			}
		}

		// function to add a node at head
		public void insertAtHead(int data) {
			Node4 newNode = new Node4(data);

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

		// function to add a node at tail
```

```
public void insertAtTail(int data) {
        Node4 newNode = new Node4(data);

        if (head == null) {
                head = newNode;
                tail = newNode;

        } else {
                tail.next = newNode;
                tail = newNode;
        }
}

// insert node between head and tail
public void insertAtPosition(int pos, int data) {
        Node4 newNode = new Node4(data);
        int totalNodes = countNodes();

        Node4 prev = null, current = head;
        // if list is empty
        if (head == null) {
                head = newNode;
                tail = newNode;
        }
        // insert at head
        else if (pos == 1) {
                newNode.next = head;
                head = newNode;
        }
        // Insert at position
        else if (pos > 1 && pos <= totalNodes + 1) {
                // insert node at tail
                if (pos == totalNodes + 1) {
                        tail.next = newNode;
                        tail = newNode;
                }
                // Insert node between head and tail
                else {
                        for (int i = 1; i < pos; i++) {
                                prev = current;
                                current = current.next;
                        }
                        newNode.next = current;
                        prev.next = newNode;
                }
        } else {
                System.out.println("Invalid node position!");
        }
}

// delete node at head
public void deleteAtHead() {
        if (head == null) {
                System.out.println("Singly linked list is empty!");
        } else {
        // If there's only one node in teh list, head and tail both become null
                if (head == tail) {
```

```
                    head = tail = null;
                } else {
                    // Otherwise, move the head pointer to the next node
                    Node4 temp = head;
                    head = head.next;
                    temp = null;
                }
        }
}
// delete node at tail
public void deleteAtTail() {
        if (head == null) {

                System.out.println("Singly linked list is empty!");
        } else if (head == tail) {
// If there's only one node in the list, head and tail both become null

                head = tail = null;
        } else {
                Node4 current = head;
                while (current.next != tail) {
                        current = current.next;
                }
                current.next = null; // Remove the last node
                tail = current; // Update the tail to the second last node
        }
}
// deleting node between head and tail
public void deleteAtPosition(int pos) {
        Node4 prev = null, current = head;
        int totalNodes = countNodes();
        // if list is empty
        if(head == null) {
                System.out.println("Singly linked list is empty");
        }
        // if head is to be deleted
        else if (pos == 1) {
                Node4 temp=head;
                head = head.next;
                if(head==null) {
                        // if there's only one node left
                        tail = null;
                }
                temp = null;
        }
        else if(pos>1 && pos <=totalNodes) {
                for(int i=1;i<pos;i++) {
                        prev = current;
                        current = current.next;
                }
                //deleting last node
                if(current.next== null) {
                        prev.next =null;
                        tail = prev;
                }
                // Delete specific node between head and tail
                else {
```

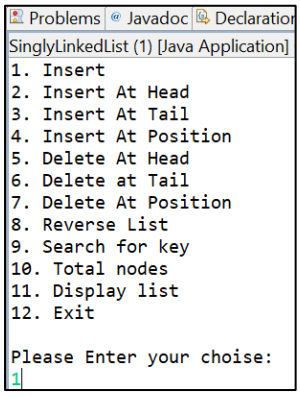
```
                        prev.next = current.next;
                    }
                }
                else {
                    System.out.println("Invalid node position!");
                }
        }
        // function to display the data in the list
        public void displayList() {
                // Pointing the head to the node called current
                Node4 current = head;

                if(head == null) {

                    System.out.println("The given list is empty!");
                    return;
                }
                System.out.println("The data in the given list are: ");


                while(current!=null) {
        // Printing each data in the list and next pointer pointing to the next node
                    System.out.print(current.data+" --> ");
                    current = current.next;
                }
                System.out.print("Null");
                System.out.println();
        }

        //Function to count total nodes
        public int countNodes() {
                int count = 0;
                Node4 current = head;
                while(current!=null) {
                    //Increment the count by 1 for each node
                    count++;
                    current = current.next;
                }
                return count;
        }
        // Reverse the node of list and print it
        public void reverseList() {
                Node4 curr = head, prev = null, temp;
                tail = curr;
                // Checks if list is empty
                if(head==null) {
                    System.out.println("list is empty");
                    return;
                }
                // Traverse all the nodes of linked list
                while(curr != null) {
                    // Store next node in temp
                    temp = curr.next;

                    // Reverse current node's next pointer
                    curr.next = prev;
```

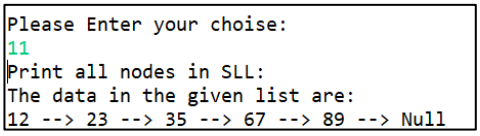
```
                // Move pointers one position ahead
                prev =  curr;
                curr = temp;
            }
            head = prev; // prev is head
        }

        // search key in linked list
        public void search(int key) {
            Node4 current = head;
            int flag=0;
            int pos = 1;
            while(current!=null) {
                if(current.data==key) {
                System.out.println("The "+key+" is found at "+pos+" position!");
                        flag=1;
                }
                current=current.next;
                pos++;
            }

            if(flag==0) {
                System.out.println(key+" not found!");
            }
        }
        public static void main(String args[]) {
            SinglyLinkedList sll = new SinglyLinkedList();
            Scanner sc = new Scanner(System.in);
            int data, pos;
            int choice;

            do {
                System.out.println("1. Insert");
                System.out.println("2. Insert At Head");

                System.out.println("3. Insert At Tail");
                System.out.println("4. Insert At Position");
                System.out.println("5. Delete At Head");
                System.out.println("6. Delete at Tail");
                System.out.println("7. Delete At Position");
                System.out.println("8. Reverse List");
                System.out.println("9. Search for key");
                System.out.println("10. Total nodes");
                System.out.println("11. Display list");
                System.out.println("12. Exit");
                System.out.println("\nPlease Enter your choise: ");
                choice = sc.nextInt();
                switch (choice) {
                case 1:
                        System.out.println("\n Enter data: ");
                        data = sc.nextInt();
                        sll.insert(data);
                        sll.displayList();
                        break;
                case 2:
                        System.out.println("\n Insert Node at head - Enter data: ");
                        data = sc.nextInt();
```

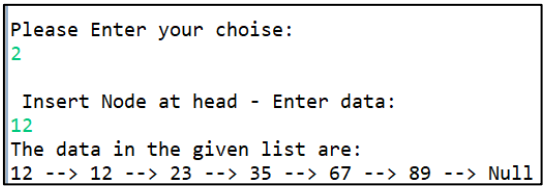
```
                sll.insertAtHead(data);
                sll.displayList();
                break;
            case 3:
                System.out.println("\n Insert Node at Tail - Enter data: ");
                data = sc.nextInt();
                sll.insertAtTail(data);
                sll.displayList();

                break;
            case 4:
                System.out.println("\n Insert Node at Position - Enter position: ");
                pos = sc.nextInt();
                System.out.println("Enter the data: ");
                data = sc.nextInt();
                sll.insertAtPosition(pos, data);
                System.out.println();
                sll.displayList();
                break;
            case 5:
                System.out.println("\n Delete node at head: ");
                sll.deleteAtHead();
                sll.displayList();
                break;
            case 6:
                System.out.println("\n Delete node at tail: ");
                sll.deleteAtTail();
                sll.displayList();
                break;
            case 7:
                System.out.println("\n Delete node at position - Enter position: ");
                pos = sc.nextInt();
                sll.deleteAtPosition(pos);
                sll.displayList();

                break;
            case 8:
                System.out.println("Reverse the SLL: ");
                sll.reverseList();
                sll.displayList();
                break;
            case 9:
                System.out.println("Enter a key to search:");
                int key = sc.nextInt();
                sll.search(key);
                break;
            case 10:
                System.out.println("Total nodes in SLL: "+sll.countNodes());
                break;
            case 11:
                System.out.println("Print all nodes in SLL: ");
                sll.displayList();
                break;
            case 12:
                System.out.println("Exiting the program.");
                break;
            default:
                System.out.println("You entered wrong choice!");
```

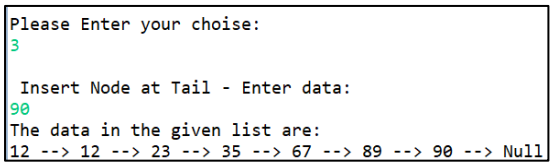
```
                    }
                } while(choice!=12);

                sc.close();

            }
        }
```

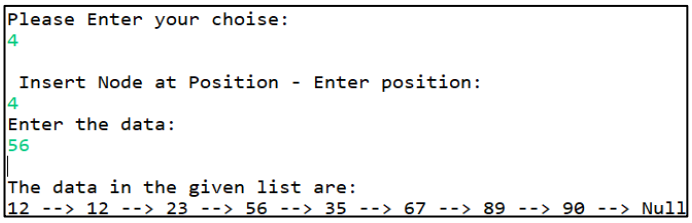
OUTPUT: -

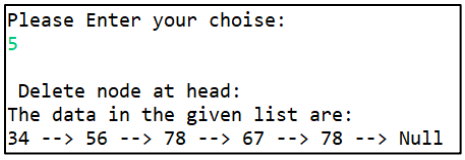


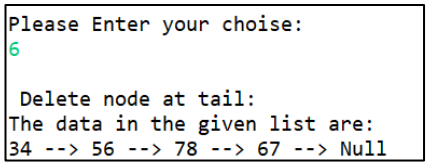


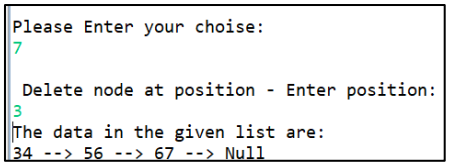


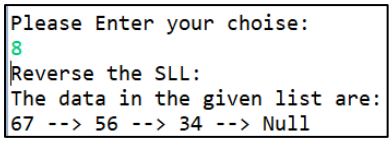


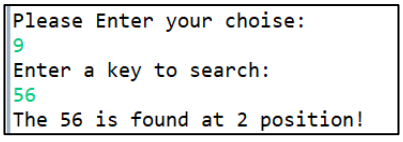


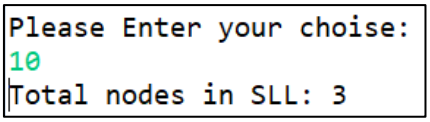


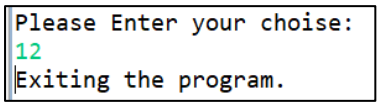












2. Write a java program to demonstrate the working of Circular Linked List.

CODE: -

```
package myStack;

//java program to implement Circular linked list
import java.util.Scanner;

class Node5 {
        int data;
        Node5 next;

        public Node5(int data) {
```

```
                this.data = data;
                this.next = null;
        }

}
public class CircularLinkedList {
        // defining the head and tail of a singly linked list
        Node5 head;
        Node5 tail;

        public CircularLinkedList() {
                head = null;

                tail = null;
        }
        public boolean isEmpty() {
                return (head == null);
        }
        // function to add a node to the list
        public void insert(int data) {
                Node5 newNode = new Node5(data);
                // if(isEmpty()){}
                if (head == null) {
                        newNode.next = newNode;
                        head = newNode;
                        tail = newNode;
                } else {
                        newNode.next = head;
                        tail.next = newNode;
                        tail = newNode;
                }
        }
        // function to add a node at head
        public void insertAtHead(int data) {

                Node5 newNode = new Node5(data);

                if (head == null) {
                        newNode.next = newNode;
                        head = newNode;
                        tail = newNode;
                } else {
                        newNode.next = head;
                        head = newNode;
                        tail.next = head;
                }
        }
        // function to add a node at tail
        public void insertAtTail(int data) {
                Node5 newNode = new Node5(data);

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

```
                tail.next = head;
            }
        }
        // insert node between head and tail
        public void insertAtPosition(int pos, int data) {
            Node5 newNode = new Node5(data);
            int totalNodes = countNodes();

            Node5 prev = null, current = head;

            // if list is empty
            if (head == null) {
                newNode.next = newNode;
                head = newNode;
                tail = newNode;
            }
            // insert at head
            else if (pos == 1) {
                tail.next = newNode;
                newNode.next = head;
                head = newNode;

            }
            // Insert at position
            else if (pos > 1 && pos <= totalNodes + 1) {
                // insert node at tail
                if (pos == totalNodes + 1) {
                    tail.next = newNode;
                    tail = newNode;
                    tail.next = head; // add
                }
                // Insert node between head and tail
                else {
                    for (int i = 1; i < pos; i++) {

                        prev = current;
                        current = current.next;
                    }
                    newNode.next = current;
                    prev.next = newNode;
                }
            } else {
                System.out.println("Invalid node position!");
            }
        }
        // delete node at head
        public void deleteAtHead() {
            if (head == null) {
                System.out.println("Circular linked list is empty!");
            } else {
                // If there's only one node in teh list, head and tail both become null
                if (head == tail) {
                    head = tail = null;
                } else {
                    // Otherwise, move the head pointer to the next node
                    Node5 temp = head;
                    head = head.next; // move head to the next node
                    tail.next = head; // maintain the circular link
```

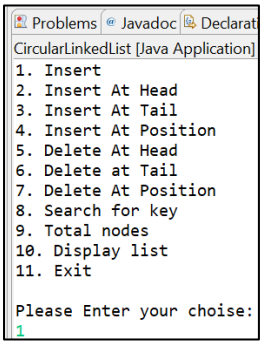
```
                        temp = null;
                }
        }
}
// delete node at tail
public void deleteAtTail() {
        if (head == null) {
                System.out.println("Circular Linked List Is Empty!");
        } else if (head == tail) {
                // If there's only one node in the list, head and tail both become null
                head = tail = null;
        } else {
                Node5 current = head;
                while (current.next != tail) {
                        current = current.next;

                }
                current.next = head; // Maintain the circular link
                tail = current; // Update the tail to the second last node
        }
}
// deleting node between head and tail
public void deleteAtPosition(int pos) {

        // if list is empty
        if(head == null) {
                System.out.println("Circular Linked List Is Empty");
        }
        // if head is to be deleted
        else if (pos == 1) {

                //single node case
                if(head==tail) {
                        // if there's only one node left

                        head = null;
                        tail = null;
                }
                else {
                        // more than one node
                        Node5 temp = head;
                        head = head.next;
                        tail.next = head; // Maintain the circular link
                        temp = null;
                }
        }
        else {
                int totalNodes = countNodes();
                if(pos>1 && pos <=totalNodes) {
                        Node5 prev = null, current = head;

                        for(int i=1;i<pos;i++) {
                                prev = current;
                                current = current.next;
                        }
                        //deleting last node
                        if(pos==totalNodes) {
                                tail = prev;
```

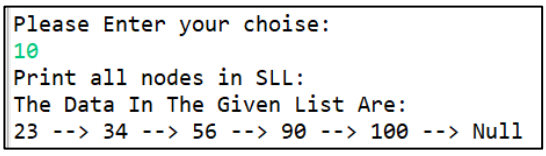
```
                                tail.next =head;// maintain the circular link
                        }
                        // Delete specific node between head and tail
                        else {
                                prev.next = current.next;
                        }
                }
                else {
                        System.out.println("Invalid Node Position!");
                }
        }
}
// function to display the data in the list
public void displayList() {
        // Pointing the head to the node called current
        Node5 current = head;

        if (head == null) {
                System.out.println("The Given List Is Empty!");
                return;
        }

        System.out.println("The Data In The Given List Are: ");

        while (true) { //change
        // Printing each data in the list and next pointer pointing to the next node
                System.out.print(current.data + " --> ");
                current = current.next;
                if(current==tail.next) break; //add
        }
        System.out.print("Null");
        System.out.println();
}
// Function to count total nodes

public int countNodes() {
        int count = 0;
        Node5 current = head;
        if (head==null) return count; //add
        do{ //change

                // Increment the count by 1 for each node
                current = current.next;
                count++;
        }while(current!=head); // change
        return count;
}


// search key in linked list
public void search(int key) {
        Node5 current = head;
        int flag = 0;
        int pos = 1;
        do{
                if (current.data == key) {
                System.out.println("The " + key + " is Found at " + pos + " Position!");
                        flag = 1;
```

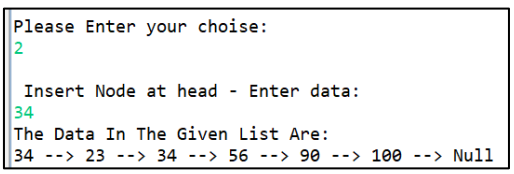
```
                }
                current = current.next;
                pos++;
        }while(current != head);

        if (flag == 0) {

                System.out.println(key + " not found!");
        }
}
public static void main(String args[]) {
        CircularLinkedList cll = new CircularLinkedList();
        Scanner sc = new Scanner(System.in);
        int data, pos;

        int choice;

        do {
                System.out.println("1. Insert");
                System.out.println("2. Insert At Head");
                System.out.println("3. Insert At Tail");
                System.out.println("4. Insert At Position");
                System.out.println("5. Delete At Head");
                System.out.println("6. Delete at Tail");
                System.out.println("7. Delete At Position");
                System.out.println("8. Search for key");
                System.out.println("9. Total nodes");
                System.out.println("10. Display list");
                System.out.println("11. Exit");
                System.out.println("\nPlease Enter your choise: ");
                choice = sc.nextInt();
                switch (choice) {
                case 1:
                        System.out.println("\nEnter data: ");
                        data = sc.nextInt();
                        cll.insert(data);
                        cll.displayList();
                        break;
                case 2:
                        System.out.println("\n Insert Node at head - Enter data: ");
                        data = sc.nextInt();
                        cll.insertAtHead(data);
                        cll.displayList();
                        break;
                case 3:
                        System.out.println("\n Insert Node at Tail - Enter data: ");
                        data = sc.nextInt();
                        cll.insertAtTail(data);
                        cll.displayList();
                        break;
                case 4:
                        System.out.println("\n Insert Node at Position - Enter position: ");
                        pos = sc.nextInt();
                        System.out.println("Enter the data: ");
                        data = sc.nextInt();
                        cll.insertAtPosition(pos, data);
                        System.out.println();
                        cll.displayList();
```

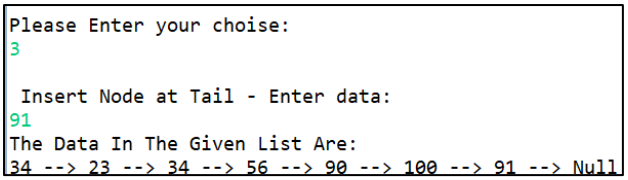
```
                    break;
                case 5:
                    System.out.println("\n Delete node at head: ");
                    cll.deleteAtHead();
                    cll.displayList();
                    break;
                case 6:
                    System.out.println("\n Delete node at tail: ");
                    cll.deleteAtTail();
                    cll.displayList();
                    break;


                case 7:
                    System.out.println("\n Delete node at position - Enter position: ");
                    pos = sc.nextInt();
                    cll.deleteAtPosition(pos);
                    cll.displayList();
                    break;

                case 8:
                    System.out.println("Enter a key to search:");
                    int key = sc.nextInt();
                    cll.search(key);
                    break;
                case 9:
                    System.out.println("Total nodes in SLL: " + cll.countNodes());
                    break;
                case 10:
                    System.out.println("Print all nodes in SLL: ");
                    cll.displayList();
                    break;
                case 11:
                    System.out.println("Exiting the program.");
                    break;
                default:
                    System.out.println("You entered wrong choice!");

                }
            } while (choice != 11);
            sc.close();

        }
}
```

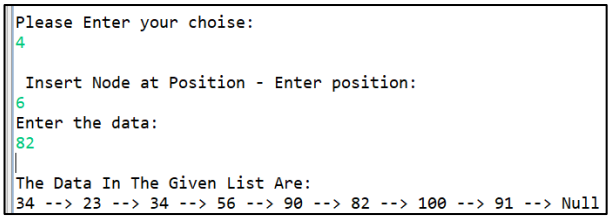
OUTPUT: -

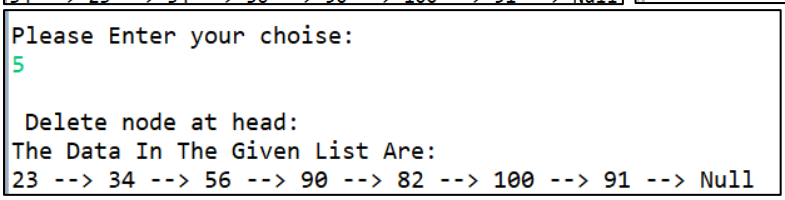


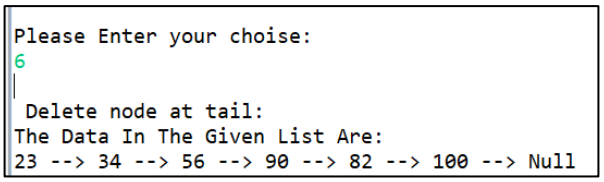


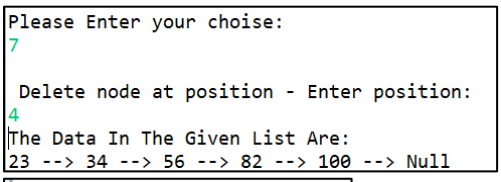


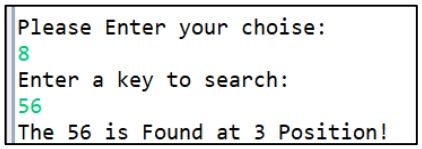


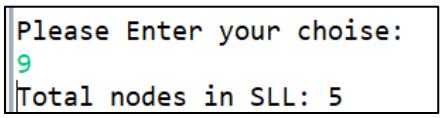


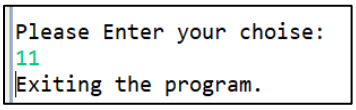












3. Write a java program to demonstrate the working of Doubly Linked List.

CODE: -

```
package doublyLinkedList;
import java.util.Scanner;
class Node {
	int data;
	Node prev; // add
	Node next;
	public Node(int data) {
		this.data = data;
		this.prev = null; // add
		this.next = null;
	}
}
public class DoublyLinkedList {
	// defining the head and tail of a singly linked list
	Node head;
	Node tail;
	public DoublyLinkedList() {
		head = null;
		tail = null;
	}
	public boolean isEmpty() {
		return (head == null);
	}
	// function to add a node to the list
	public void insert(int data) {
		Node newNode = new Node(data);
		// if(isEmpty()){}
		if (head == null) {
```

```
                head = newNode;
                tail = newNode;
            } else {
                tail.next = newNode;
                newNode.prev = tail; // add
                tail = newNode;
            }
        }
        // function to add a node at head
        public void insertAtHead(int data) {
            Node newNode = new Node(data);
            if (head == null) {

                head = newNode;
                tail = newNode;
            } else {
                newNode.next = head;
                head.prev = newNode; // add
                head = newNode;
            }
        }
        // function to add a node at tail

        public void insertAtTail(int data) {
            Node newNode = new Node(data);
            if (head == null) {
                head = newNode;
                tail = newNode;
            } else {
                tail.next = newNode;
                newNode.prev = tail;
                tail = newNode;
            }
        }
        // insert node between head and tail
        public void insertAtPosition(int pos, int data) {
            Node newNode = new Node(data);
            // if list is empty
            if (head == null) {
                head = newNode;
                tail = newNode;
            }
            // insert at head
            else if (pos == 1) {
                newNode.next = head;
                head.prev = newNode;
                head = newNode;
            }
            // Insert at position
            else {
//                  if (pos > 1 && pos <= totalNodes + 1) remove this condition from else
                Node current = head;
                int currPos = 1;
                while (current != null && currPos < pos) {
```

```
                current = current.next;
                currPos++;
            }
            if (pos < 1 || pos > currPos) {
                System.out.println("Invalid node position!");
                return;
            }
            // insert node at tail
            if (current == null) {
                if (tail == null) {
                    head = newNode;
                    tail = newNode;

                } else {
                    tail.next = newNode;
                    newNode.prev = tail;
                    tail = newNode;
                }
            } // insert at middle
            else {
                newNode.next = current;
                newNode.prev = current.prev;
                current.prev.next = newNode;

                current.prev = newNode;
            }
        }
    }
    // delete node at head
    public void deleteAtHead() {
        if (head == null) {
            System.out.println("doubly linked list is empty!");
        } else if (head == tail) {
            head = tail = null;
        }
        // If there's only one node in the list, head and tail both become null
        else {
            // Otherwise, move the head pointer to the next node
            Node temp = head;
            head = head.next; // move head to the next node
            head.prev = null; // maintain the circular link
            temp = null;
        }
    }
    // delete node at tail
    public void deleteAtTail() {
        if (head == null) {
            System.out.println("doubly linked list is empty!");
        } else if (head == tail) {
            // If there's only one node in the list, head and tail both become null
            head = tail = null;
        } else {
            Node temp = tail;
            tail = tail.prev;
```

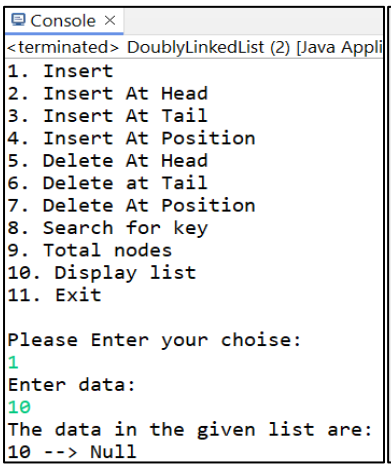
```
                tail.next = null;
                temp.prev = null;
        }
}
// deleting node at head or tail or between head and tail
public void deleteAtPosition(int pos) {
        Node current = head;
        int totalNodes = countNodes();
        // if list is empty
        if (head == null) {
                System.out.println("Doubly linked list is empty");
        }

        // if head is to be deleted
        else if (pos == 1) {
                // single node case
                if (head == tail) {
                        // if there's only one node left
                        head = null;
                        tail = null;
                } else {
                        // more than one node
                        Node temp = head;
                        head = head.next;

                        head.prev = null; // add
                        temp.next = null; // add
                }
        } else if (pos > 1 && pos <= totalNodes + 1) { // delete node between head and tail
                if (pos == totalNodes) {
                        // if only one node
                        if (head.next == null) {
                                head = null;
                                tail = null;
                        }
                        // if one or more nodes
                        else {
                                Node temp = tail;
                                tail = tail.prev;
                                tail.next = null;
                                temp.prev = null;
                        }
                } else {
                        for (int i = 1; i < pos; i++) {
                                current = current.next;
                        }
                        current.prev.next = current.next; // add
                        current.next.prev = current.prev; // add
                        current.prev = null; // add
                        current.next = null; // add
                }
        } else {
                System.out.println("Invalid node position!");
        }
```

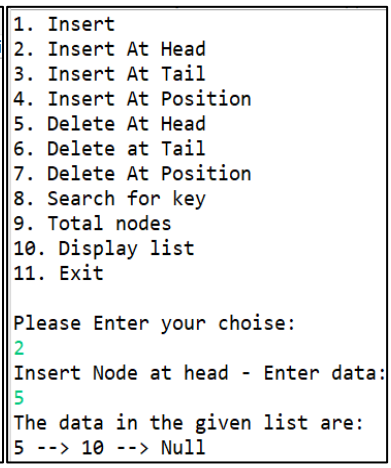
```
}
// function to display the data in the list
public void displayList() {
	// Pointing the head to the node called current
	Node current = head;
	if (head == null) {
		System.out.println("The given list is empty!");
		return;
	}
	System.out.println("The data in the given list are: ");
	while (current != null) { // change
		// Printing each data in the list and next pointer pointing to the next node

		System.out.print(current.data + " --> ");
		current = current.next;
	}
	System.out.print("Null");
	System.out.println();
}
// Function to count total nodes
public int countNodes() {
	int count = 0;
	Node current = head;
	if (head == null)
		return count; // add

	while (current != null) { // change
		// Increment the count by 1 for each node
		count++;
		current = current.next;
	} // change
	return count;
}

// search key in linked list
public void search(int key) {
	Node current = head;
	int flag = 0;
	int pos = 1;
	while (current != null) {
		if (current.data == key) {
			System.out.println("The " + key + " is found at " + pos + " position!");
			flag = 1;
		}
		current = current.next;
		pos++;
	}

	if (flag == 0) {
		System.out.println(key + " not found!");
	}
}
public static void main(String args[]) {
	DoublyLinkedList dll = new DoublyLinkedList();
```

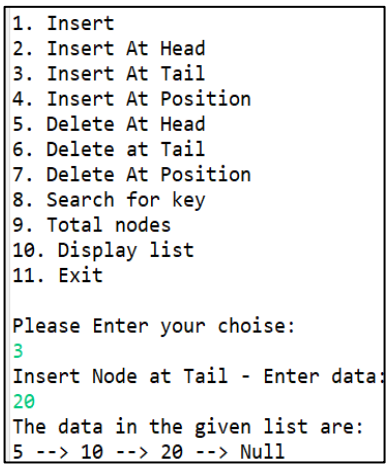
```
Scanner sc = new Scanner(System.in);
int data, pos;
int choice;
do {
        System.out.println("1. Insert");
        System.out.println("2. Insert At Head");
        System.out.println("3. Insert At Tail");
        System.out.println("4. Insert At Position");
        System.out.println("5. Delete At Head");
        System.out.println("6. Delete at Tail");
        System.out.println("7. Delete At Position");
        System.out.println("8. Search for key");
        System.out.println("9. Total nodes");
        System.out.println("10. Display list");
        System.out.println("11. Exit");
        System.out.println("\nPlease Enter your choise: ");
        choice = sc.nextInt();
        switch (choice) {
        case 1:
                System.out.println("Enter data: ");
                data = sc.nextInt();
                dll.insert(data);
                dll.displayList();
                break;
        case 2:

                System.out.println("Insert Node at head - Enter data: ");
                data = sc.nextInt();
                dll.insertAtHead(data);
                dll.displayList();
                break;
        case 3:
                System.out.println("Insert Node at Tail - Enter data: ");
                data = sc.nextInt();
                dll.insertAtTail(data);
                dll.displayList();
                break;
        case 4:
                System.out.println("Insert Node at Position - Enter position: ");
                pos = sc.nextInt();
                System.out.println("Enter the data: ");
                data = sc.nextInt();
                dll.insertAtPosition(pos, data);
                System.out.println();
                dll.displayList();
                break;
        case 5:
                System.out.println("Delete node at head: ");
                dll.deleteAtHead();
                dll.displayList();
                break;
        case 6:
                System.out.println("Delete node at tail: ");
                dll.deleteAtTail();
```

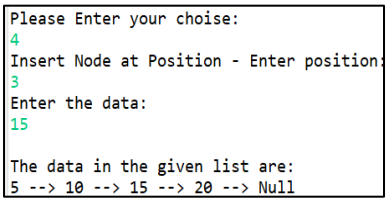
```
                    dll.displayList();
                    break;
                case 7:
                    System.out.println("Delete node at position - Enter position: ");
                    pos = sc.nextInt();
                    dll.deleteAtPosition(pos);
                    dll.displayList();
                    break;
                case 8:
                    System.out.println("Enter a key to search:");
                    int key = sc.nextInt();

                    dll.search(key);
                    break;
                case 9:
                    System.out.println("Total nodes in DLL: " + dll.countNodes());
                    break;
                case 10:
                    System.out.println("Print all nodes in DLL: ");
                    dll.displayList();
                    break;
                case 11:
                    System.out.println("Exiting the program.");
                    break;
                default:
                    System.out.println("You entered wrong choice!");
                }
            } while (choice != 11);
            sc.close();

        }
}
```

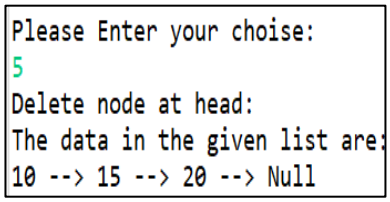
OUTPUT: -

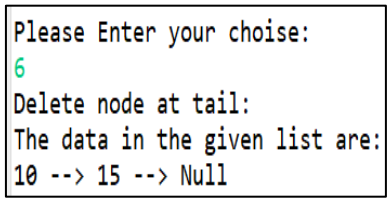












```
Please Enter your choise:
1
Enter data:
20
The data in the given list are:
10 --> 15 --> 20 --> Null
```

```
Please Enter your choise:
1
Enter data:
40
The data in the given list are:
10 --> 15 --> 20 --> 40 --> Null
```

```
Please Enter your choise:
6
Delete node at tail:
The data in the given list are:
10 --> 15 --> 20 --> Null
```

```
Please Enter your choise:
7
Delete node at position - Enter position:
2
The data in the given list are:
10 --> 20 --> Null
```

```
Please Enter your choise:
8
Enter a key to search:
10
The 10 is found at 1 position!
```

```
Please Enter your choise:
8
Enter a key to search:
90
90 not found!
```

```
Please Enter your choise:
10
Print all nodes in DLL:
The data in the given list are:
10 --> 20 --> Null
```

```
Please Enter your choise:
11
Exiting the program.
```

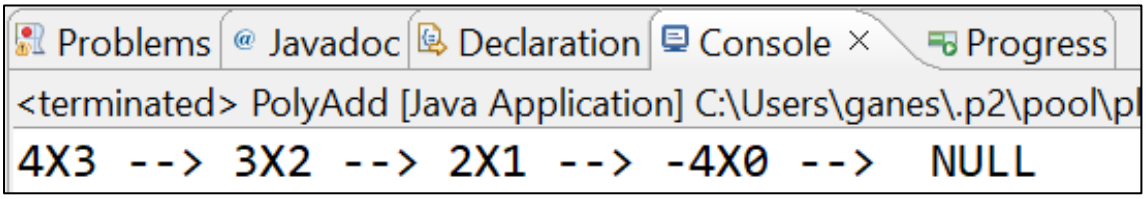
4. Write a java program to perform addition of two polynomials using Linked List.

CODE: -

```
package myPoly;

//addition of polynomial Equation using LinkedList
class Node3{
	int coeff;
	int pow;
	Node3 next;

	public Node3(int c,int p) {
		coeff = c;
		pow = p;
		next = null;
	}
}
public class PolyAdd {
	static Node3 addPolynomial(Node3 head1, Node3 head2) {
		if(head1 == null)
			return head2;
		if(head2 == null)
			return head1;

		if(head1.pow > head2.pow) {
			Node3 nextPtr = addPolynomial(head1.next, head2);
			head1.next = nextPtr;
			return head1;
		}
		else if(head1.pow<head2.pow) {
			Node3 nextPtr = addPolynomial(head1, head2.next);
			head2.next = nextPtr;
			return head2;
		}
		Node3 nextPtr = addPolynomial(head1.next, head2.next);
```

```
                head1.coeff = head1.coeff + head2.coeff;
                head1.next = nextPtr;
                return head1;
        }
        static void printPloynomialAddition(Node3 head) {
                Node3 current = head;

                while(current != null) {
                        System.out.print(current.coeff+"X"+ current.pow+" --> ");

                        current = current.next;
                }
                System.out.print(" NULL");

        }
        public static void main(String args[]) {
                //create first polynomial equation
                Node3 head1 = new Node3(4, 3);
                head1.next = new Node3(3, 2);
                head1.next.next = new Node3(3 , 0);

                Node3 head2 = new Node3(2 , 1);
                head2.next = new Node3(-7, 0);

                Node3 head = addPolynomial(head1, head2);
                printPloynomialAddition(head);
        }
}
```

OUTPUT: -



5. **Write a java program to demonstrate the working of Stack using Linked List.**

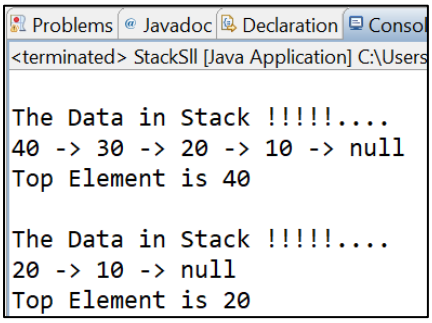
CODE: -

```
package myStack;

class Node{
        //java program to implements stack using Singly linked list
        int data;

        Node next;

        Node(int d) {
                this.data = d;
                this.next = null;
        }
}
class Stack{
```

```
	Node head;

	Stack() {
		this.head = null;
	}
	Boolean isEmpty(){
		return head == null;
	}
	void push(int d) {
		Node newNode = new Node(d);
		if (newNode == null) {
		System.out.println("\nStack is OverFlow !!!!!!.....");
			return ;
		}
		newNode.next = head;
		head = newNode;
	}
	void pop() {
		if(isEmpty()) {
			System.out.println("\nStack is UnderFlow !!!!....");
			return ;
		}else {
			Node temp = head;
			head = head.next;
			temp = null;
		}
	}
	int peek() {
	   if (!isEmpty()) { // Check if the stack is NOT empty
	      return head.data;
	   } else {
	      System.out.println("\nStack is Empty !!!!.....");
	      return Integer.MIN_VALUE;
	   }
	}
	public void displayStack() {
		Node current = head;
		if(head == null) {
			System.out.println("\nStack is Empty !!!!......");
			return ;
		}
		System.out.println("\nThe Data in Stack !!!!!....");
		while(current != null) {
			System.out.print(current.data+" -> ");

			current = current.next;

		}
		System.out.print("null");
		System.out.println();
	}
}
public class StackSll {
	public static void main(String args []) {
		Stack st = new Stack();

		st.push(10);
		st.push(20);
```

```
                st.push(30);
                st.push(40);
                st.displayStack();
                System.out.println("Top Element is "+st.peek());
                st.pop();
                st.pop();
                st.displayStack();
                System.out.println("Top Element is "+st.peek());

        }

}
```

OUTPUT: -



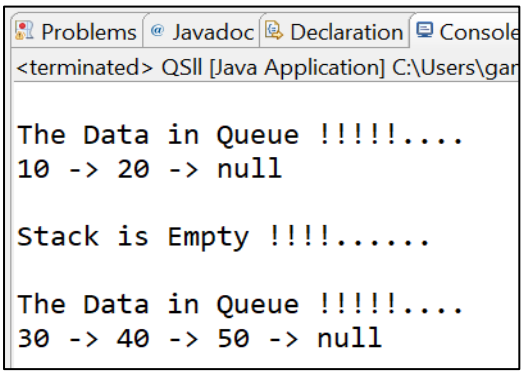
6. **Write a java program to demonstrate the working of Queue using Linked List.**

CODE: -

```
package myQueue;

class Node1{
        //java program to implements queue data structure using linked list
        int data;
        Node1 next;

        Node1(int d) {
                this.data = d;
                this.next = null;
        }
}
class Queue{
        Node1 front, rear;

        Queue(){
                front = rear = null;
        }
        boolean isEmpty() {
                return front == null && rear == null;
        }
        void enqueue(int d) {
                Node1 new_node = new Node1(d);
                if(rear == null) {
                        front = rear = new_node;
                        return ;
                }
                rear.next = new_node;
```

```
                rear = new_node;
        }
        void dequeue() {
                if(isEmpty()) {
                        System.out.println("Queue UnderFlow!!!!");
                        return ;
                }
                Node1 temp = front;
                front = front.next;
                if(front == null) {

                        rear = null;
                }
        }
        public void displayQueue() {
                Node1 current = front;
                if(front == null) {
                        System.out.println("\nStack is Empty !!!!.....");
                        return ;
                }
                System.out.println("\nThe Data in Queue !!!!!....");
                while(current != null) {
                        System.out.print(current.data+" -> ");
                        current = current.next;
                }
                System.out.print("null");
                System.out.println();
        }
}
public class QSll {
        public static void main(String[] args) {

                // TODO Auto-generated method stub
                Queue q = new Queue();

                //Enqueue element into the queue
                q.enqueue(10);
                q.enqueue(20);

                q.displayQueue();

                q.dequeue();
                q.dequeue();

                q.displayQueue();

                q.enqueue(30);
                q.enqueue(40);
                q.enqueue(50);

                q.displayQueue();
        }
}
```

OUTPUT: -



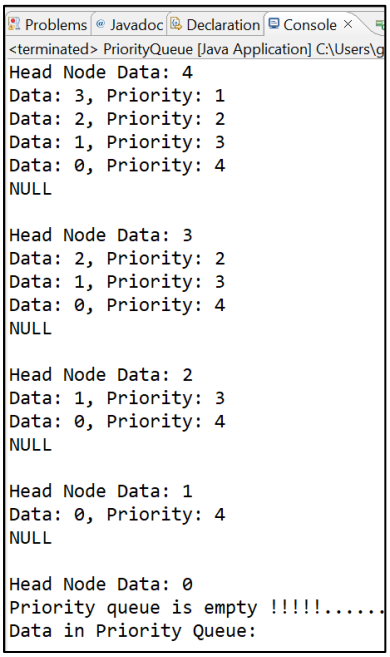
7. **Write a java program to demonstrate the working of Priority Queue using Linked List.**

CODE: -

```
package myPQueue;

class Node4{
	Node4 next;
	int data;
	int priority;//lower values indicate =(lower value= higher priority)

	public Node4(int d, int p) {
		this.data = d;
		this.priority = p;
		this.next = null;
	}
}
public class PriorityQueue {
	Node4 head;

	public PriorityQueue() {


		head = null;
	}
	public void add(int d, int p) {
		Node4 start = head;
		Node4 newNode = new Node4(d, p);
		if(head == null) {
			head = newNode;
			return ;
		}
		if(head.priority > p) {
			newNode.next = head;
			head = newNode;
		}else {
			 // Case 2: Traverse and find the correct position to insert the new node
			while (start.next != null && start.next.priority <= p) {
				start = start.next;
			}
			newNode.next = start.next;
			start.next = newNode;
		}
```

```
}
        // Method to remove the node with the highest priority (lowest priority value)
        public Node4 remove() {
                Node4 temp = head;
                head = head.next;
                temp = null;
                return head;
        }
        int getHeadData() {
                return head.data;
        }

        public boolean isEmpty() {
                return head == null;// Return true if the queue is empty
        }

        public void display() {
                if(head == null) {
                        System.out.println("Priority queue is empty !!!!!.....");
                        System.out.println("Data in Priority Queue: ");
                        return;

                }
                Node4 start = head;
                while(start != null) {
                        System.out.println("Data: " + start.data + ", Priority: " + start.priority);
                        start = start.next;
                }
                System.out.println("NULL\n");
}
public static void main (String args []) {
                PriorityQueue pq = new PriorityQueue();
                pq.add(0, 4); //(data: 0, priority: 4)
                pq.add(1, 3);
                pq.add(2, 2);
                pq.add(3, 1);
                pq.add(4, 0);

                //System.out.println("Head Node Data: "+pq.getHeadData());

                while(!pq.isEmpty()) {
                        System.out.println("Head Node Data: "+pq.getHeadData());

                        pq.remove();
                        pq.display();
                }
        }
}
```

OUTPUT: -



8. **Write a java program to demonstrate the working of Double Ended Queue using Linked List.**

CODE: -

```
package myPQueue;

import java.util.Scanner;

class Node{
	int data;
	Node prev;
	Node next;

	// Function to get a new node

	public Node(int data) {

		this.data = data;
		this.prev = null;
		this.next = null;
	}
}
public class DEQueue {

	Node front;
	Node rear;
	int Size;

	public DEQueue() {
		front = rear = null;
		Size = 0;
	}

	public boolean isEmpty() {
```

```
        return (front==null);
    }

    // return the number of elements in the deque
    public int size() {
        return Size;
    }

    // insert an element at the front end
    public void insertFront(int data) {
        Node newNode = new Node(data);
        if(newNode==null) {
            System.out.println("OverFlow!\n");
        }
        else {
            // if deque is empty
            if(front == null) {
                //insert node at the front end
                rear = front = newNode;

            }
            else {
                newNode.next = front;
                front.prev = newNode;
                front = newNode;
            }
            Size++; // to count element
        }
    }
    // insert an element at the rear end
    public void insertRear(int data) {
        Node newNode = new Node(data);
        if(newNode==null) {
            System.out.println("Overflow!\n");
        }
        else {
            // if deque is empty
            if(rear==null) {

                // insert node at the rear end
                front = rear = newNode;
            }
            else {
                newNode.prev = rear;
                rear.next = newNode;
                rear = newNode;
            }
            Size++;
        }
    }
    // delete the element from the front end
    public void deleteFront() {
        // if deque is empty then underflow condition
        if(isEmpty()) {
            System.out.println("Underflow - deque is empty");
        }
        //deletes the node form the front end
        else {
```

```
                Node temp = front;
                front = front.next;
                // if only one element was present
                if(front ==null) {
                        rear=null;
                }
                else {
                        front.prev = null;
                }
                // decrements the count of elements by 1
                Size--;
        }
}

// delete the element from the rear end
void deleteRear() {
        // if deque is empty then 'underflow' condition
        if(isEmpty()) {
                System.out.println("Underflow - deque is empty!\n");

        }
        // deletes the node from the rear end
        else {
                Node temp = rear;
                rear = rear.prev;

                // if only one elements was present
                if(rear==null) {
                        front=null;
                }
                else {
                        rear.next = null;
                }
                //Decrements count of elements by 1
                Size--;
        }
}
// return the elements at the front end
public int getFront() {
        // if deque is empty, then return -1 value
        if(isEmpty()) {
                return -1;
        }
        return front.data;
}
// return the elements at the rear end
public int getRear() {
        // if deque is empty, then return -1 value
        if(isEmpty()) {
                return -1;
        }
        return rear.data;

}
public void display() {
        Node current = front;
        if(front==null) {
                System.out.println("The double ended queue is empty!");
```

```
                return;
        }
        System.out.println("The data in double ended queue are: ");
        System.out.print("Null <- ");
        while(current!=null) {
                System.out.print(current.data+" <-> ");
                current = current.next;
        }
        System.out.print("Null");
        System.out.println();
}

public static void main(String[] args) {
        DEQueue deq = new DEQueue();
        Scanner sc = new Scanner(System.in);
        int data, pos;
        int choice;

        do {

                System.out.println("1. Insert Front");
                System.out.println("2. Insert Rear");
                System.out.println("3. Delete Front");
                System.out.println("4. Delete Rear");
                System.out.println("5. Display");
                System.out.println("6. Get Front");
                System.out.println("7. Get Rear");
                System.out.println("8. Exit");
                System.out.println("\nPlease Enter your choise: ");
                choice = sc.nextInt();
                switch (choice) {
                case 1:
                        System.out.println("\n Insert Node at front - Enter data: ");
                        data = sc.nextInt();
                        deq.insertFront(data);

                        deq.display();
                        break;
                case 2:
                        System.out.println("\n Insert Node at rear - Enter data: ");
                        data = sc.nextInt();

                        deq.insertRear(data);
                        deq.display();
                        break;
                case 3:
                        System.out.println("\n Delete node at front: ");
                        deq.deleteFront();
                        deq.display();
                        break;
                case 4:
                        System.out.println("\n Delete node at Rear: ");
                        deq.deleteRear();
                        deq.display();
                        break;
                case 5:
                        System.out.println("\n Display Nodes: ");
                        deq.display();
```

```
                    break;

                case 6:
                    System.out.println("\n The front element is : "+deq.getFront());
                    break;
                case 7:
                    System.out.println("\n The front element is : "+deq.getRear());
                    break;
                case 8:
                    System.out.println("Exiting the program.");
                    break;
                default:
                    System.out.println("You entered wrong choice!");
                }
            } while (choice != 8);
            sc.close();
        }

}
```

OUTPUT: -

```
Problems  Javadoc  Declaration  Co
DEQueue [Java Application] C:\Users\ganes\.
1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Display
6. Get Front
7. Get Rear
8. Exit

Please Enter your choise:
1
```

```
Please Enter your choise:
5

 Display Nodes:
The data in double ended queue are:
Null <- 69 <-> 49 <-> 90 <-> 67 <-> 45 <-> 23 <-> 42 <-> 78 <-> 98 <-> 76 <-> Null
```

```
Please Enter your choise:
3

 Delete node at front:
The data in double ended queue are:
Null <- 49 <-> 90 <-> 67 <-> 45 <-> 23 <-> 42 <-> 78 <-> 98 <-> 76 <-> Null
```

```
Please Enter your choise:
4

 Delete node at Rear:
The data in double ended queue are:
Null <- 49 <-> 90 <-> 67 <-> 45 <-> 23 <-> 42 <-> 78 <-> 98 <-> Null
```

```
Please Enter your choise:
6

 The front element is : 49
```

```
Please Enter your choise:
7

 The front element is : 98
```

```
Please Enter your choise:
8
Exiting the program.
```