DSA Assignment # 1

Submitted From:	**Rida Ashraf**
Submitted To:	**Sir Waqas Asif**
Course:	**(DSA)**
Section:	**SE-22-A**
Semester:	**3rd (22 Batch)**
Date:	**11/22/2023**
Roll No:	**22014198-070**

Question Statement:

Write java code to implement the following task in one program using switch statement.

1. Write code to implement Link List with mentioned functions

1. Single link list (insert at start , random , end : delete at start, random, end : search : print)
2. Circular link list (insert , print)
3. Double Link list (insert , print)

2. Write code to implement Stack operations (Push, Pop, isFull , isEmpty, Peek, Print)

1. Stack using Array
2. Stack using Linked List

3. Write code to implement Queue functions (Enqueue, Dequeue, isEmpty, is Full, getFront, getRear, Print)

1. Queue using Array
2. Queue using Linked List
3. Circular Queue using Array
4. Double ended queue (Enqueue at front , Enqueue at Rear, Dequeue at front , Dequeue at Rear , Print)

Solution:

Main.java

```java
import java.util.*; // For Input

// Linked list node
class Node {
    int data;
    Node next;
    Node prev; // For double linked list

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

// Singly linked list
class LinkedList {
    private Node head;

    LinkedList() {
    this.head = null;22
```

```
    }

    void insertAtStart(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
    head = newNode;28
    }

    void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }

        Node temp = head;
        while (temp.next != null) {
        temp = temp.next;40
        }

        temp.next = newNode;
    }

    void deleteAtFirst() {
        if (head == null) {
            System.out.println("Linkedlist is Empty...");
            return;
        }

        Node currentNode = head;
    head = currentNode.next;53
    }
```

```
54
55      void deleteAtEnd() {
56          if (head == null || head.next == null) {
57              head = null;
58              return;
59          }
60
61          Node temp = head;
62          while (temp.next.next != null) {
63          temp = temp.next;64
            }
65
66          temp.next = null;
67      }
68
69      public int search(int target) {
70          Node temp = head;
71          int i = 0;
72          while (temp != null) {
73              if (temp.data == target) {
74                  return i;
75              }
76              temp = temp.next;
77              i++;
78          }
79          return -1;
80      }
81
82      void insertRandom(int data, int position) {
83          Node newNode = new Node(data);
84          if (position < 0) {
85              System.out.println("Invalid position for insertion");
86              return;
87          }
88
89          if (position == 0) {
90              newNode.next = head;
91              head = newNode;
92              return;
93          }
94
95          Node temp = head;
96          for (int i = 0; i < position - 1 && temp != null; i++) {
97          temp = temp.next;98
            }
99
100         if (temp == null) {
101             System.out.println("Invalid position for insertion");
102             return;
103         }
104
105         newNode.next = temp.next;
106         temp.next = newNode;
107     }
108
109     void deleteRandom(int position) {
```

```
110         if (head == null || position < 0) {
111             System.out.println("Invalid position for deletion");
112             return;
113         }
114 
115         if (position == 0) {
116             if (head.next == head) {
117                 head = null; // Update head if the last element is deleted
118             } else {
119                 Node temp = head;
120                 while (temp.next != head) {
121                     temp = temp.next;
122                 }
123                 head = head.next;
124                 temp.next = head;
125             }
126             return;
127         }
128 
129         Node temp = head;
130         for (int i = 0; i < position - 1 && temp != null; i++) {
131             temp = temp.next;
132         }
133 
134         if (temp == null || temp.next == null) {
135             System.out.println("Invalid position for deletion");
136             return;
137         }
138 
139         temp.next = temp.next.next;
140 
141         if (temp.next == head) {
142             head = temp.next.next;
143         }
144     }
145 
146     void print() {
147         Node temp = head;
148         while (temp != null) {
149             System.out.print(temp.data + " -> ");
150             temp = temp.next;
151         }
152         System.out.println("Null");
153     }
154 }
155 
156 // Circular linked list
157 class CircularLinkedList {
158     private Node head;
159 
160     CircularLinkedList() {
161         this.head = null;
162     }
163 
164     void insert(int data) {
165         Node newNode = new Node(data);
```

```
166         if (head == null) {
167             head = newNode;
168             head.next = head;
169         } else {
170             Node temp = head;
171             while (temp.next != head) {
172                 temp = temp.next;
173             }
174             newNode.next = head;
175             temp.next = newNode;
176         }
177     }
178 
179     void print() {
180         if (head == null) {
181             System.out.println("Circular Linked List is empty");
182             return;
183         }
184         Node temp = head;
185         do {
186             System.out.print(temp.data + " ");
187             temp = temp.next;
188         } while (temp != head);
189         System.out.println();
190     }
191 }
192 
193 // Doubly linked list
194 class DoublyLinkedList {
195     private Node head;
196 
197     DoublyLinkedList() {
198         this.head = null;
199     }
200 
201     void insert(int data) {
202         Node newNode = new Node(data);
203         if (head == null) {
204             head = newNode;
205         } else {
206             newNode.next = head;
207             head.prev = newNode;
208             head = newNode;
209         }
210     }
211 
212     void print() {
213         Node temp = head;
214         while (temp != null) {
215             System.out.print(temp.data + " -> ");
216             temp = temp.next;
217         }
218         System.out.println("Null");
219     }
220 }
221 
```

```java
// Stack using Array
class StackArray {
    private int top;
    private int maxSize;
    private int[] stackArray;

    StackArray(int maxSize) {
        this.maxSize = maxSize;
        this.stackArray = new int[maxSize];
        this.top = -1;
    }

    boolean isEmpty() {
        return top == -1;
    }

    boolean isFull() {
        return top == maxSize - 1;
    }

    void push(int data) {
        if (!isFull()) {
            stackArray[++top] = data;
        } else {
            System.out.println("Stack overflow");
        }
    }

    int pop() {
        if (!isEmpty()) {
            return stackArray[top--];
        } else {
            System.out.println("Stack underflow");
            return -1;
        }
    }

    int peek() {
        if (!isEmpty()) {
            return stackArray[top];
        } else {
            System.out.println("Stack is empty");
            return -1;
        }
    }

    void print() {
        if (!isEmpty()) {
            for (int i = 0; i <= top; i++) {
                System.out.print(stackArray[i] + " ");
            }
            System.out.println();
        } else {
            System.out.println("Stack is empty");
        }
    }
```

```java
}

// Stack using Linked List
class StackLinkedList {
    private Node top;

    StackLinkedList() {
        this.top = null;
    }

    boolean isEmpty() {
        return top == null;
    }

    void push(int data) {
        Node newNode = new Node(data);
        newNode.next = top;
        top = newNode;
    }

    int pop() {
        if (!isEmpty()) {
            int data = top.data;
            top = top.next;
            return data;
        } else {
            System.out.println("Stack underflow");
            return -1;
        }
    }

    int peek() {
        if (!isEmpty()) {
            return top.data;
        } else {
            System.out.println("Stack is empty");
            return -1;
        }
    }

    void print() {
        Node temp = top;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}

// Queue using Array
class QueueArray {
    private int front, rear, size;
    private int capacity;
    private int[] array;

```

```
334     QueueArray(int capacity) {
335         this.capacity = capacity;
336         this.front = this.size = 0;
337         this.rear = capacity - 1;
338         this.array = new int[capacity];
339     }
340 
341     boolean isFull() {
342         return this.size == this.capacity;
343     }
344 
345     boolean isEmpty() {
346         return this.size == 0;
347     }
348 
349     void enqueue(int item) {
350         if (isFull()) {
351             System.out.println("Queue is full");
352             return;
353         }
354         this.rear = (this.rear + 1) % this.capacity;
355         this.array[this.rear] = item;
356         this.size = this.size + 1;
357     }
358 
359     int dequeue() {
360         if (isEmpty()) {
361             System.out.println("Queue is empty");
362             return -1;
363         }
364         int item = this.array[this.front];
365         this.front = (this.front + 1) % this.capacity;
366         this.size = this.size - 1;
367         return item;
368     }
369 
370     int getFront() {
371         if (isEmpty()) {
372             System.out.println("Queue is empty");
373             return -1;
374         }
375         return this.array[this.front];
376     }
377 
378     int getRear() {
379         if (isEmpty()) {
380             System.out.println("Queue is empty");
381             return -1;
382         }
383         return this.array[this.rear];
384     }
385 
386     void print() {
387         if (isEmpty()) {
388             System.out.println("Queue is empty");
389             return;
```

```
390         }
391         for (int i = 0; i < size; i++) {
392             int index = (front + i) % capacity;
393             System.out.print(array[index] + " ");
394         }
395         System.out.println();
396     }
397 }
398 
399 // Queue using Linked List
400 class QueueLinkedList {
401     private Node front, rear;
402     private int size;
403 
404     QueueLinkedList() {
405         this.front = this.rear = null;
406         this.size = 0;
407     }
408 
409     boolean isEmpty() {
410         return size == 0;
411     }
412 
413     void enqueue(int item) {
414         Node newNode = new Node(item);
415         if (isEmpty()) {
416             front = rear = newNode;
417         } else {
418             rear.next = newNode;
419             rear = newNode;
420         }
421         size++;
422     }
423 
424     int dequeue() {
425         if (isEmpty()) {
426             System.out.println("Queue is empty");
427             return -1;
428         }
429         int item = front.data;
430         front = front.next;
431         size--;
432         return item;
433     }
434 
435     int getFront() {
436         if (isEmpty()) {
437             System.out.println("Queue is empty");
438             return -1;
439         }
440         return front.data;
441     }
442 
443     int getRear() {
444         if (isEmpty()) {
445             System.out.println("Queue is empty");
```

```
            return -1;
        }
        return rear.data;
    }

    void print() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return;
        }
        Node temp = front;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}

class CircularQueue {
    private int front, rear, size;
    private int capacity;
    private int[] array;

    CircularQueue(int capacity) {
        this.capacity = capacity;
        this.front = this.size = 0;
        this.rear = capacity - 1;
        this.array = new int[capacity];
    }

    boolean isFull() {
        return this.size == this.capacity;
    }

    boolean isEmpty() {
        return this.size == 0;
    }

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

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

```
502         this.size = this.size - 1;
503         return item;
504     }
505 
506     int getFront() {
507         if (isEmpty()) {
508             System.out.println("Queue is empty");
509             return -1;
510         }
511         return this.array[this.front];
512     }
513 
514     int getRear() {
515         if (isEmpty()) {
516             System.out.println("Queue is empty");
517             return -1;
518         }
519         return this.array[this.rear];
520     }
521 
522     void print() {
523         if (isEmpty()) {
524             System.out.println("Queue is empty");
525             return;
526         }
527         for (int i = 0; i < size; i++) {
528             int index = (front + i) % capacity;
529             System.out.print(array[index] + " ");
530         }
531         System.out.println();
532     }
533 }
534 
535 class Deque {
536     private int front, rear, size;
537     private int capacity;
538     private int[] array;
539 
540     Deque(int capacity) {
541         this.capacity = capacity;
542         this.front = this.size = 0;
543         this.rear = capacity - 1;
544         this.array = new int[capacity];
545     }
546 
547     boolean isFull() {
548         return this.size == this.capacity;
549     }
550 
551     boolean isEmpty() {
552         return this.size == 0;
553     }
554 
555     void enqueueFront(int item) {
556         if (isFull()) {
557             System.out.println("Deque is full");
```

```
            return;
        }
        this.front = (this.front - 1 + this.capacity) % this.capacity;
        this.array[this.front] = item;
        this.size = this.size + 1;
    }

    void enqueueRear(int item) {
        if (isFull()) {
            System.out.println("Deque is full");
            return;
        }
        this.rear = (this.rear + 1) % this.capacity;
        this.array[this.rear] = item;
        this.size = this.size + 1;
    }

    int dequeueFront() {
        if (isEmpty()) {
            System.out.println("Deque is empty");
            return -1;
        }
        int item = this.array[this.front];
        this.front = (this.front + 1) % this.capacity;
        this.size = this.size - 1;
        return item;
    }

    int dequeueRear() {
        if (isEmpty()) {
            System.out.println("Deque is empty");
            return -1;
        }
        int item = this.array[this.rear];
        this.rear = (this.rear - 1 + this.capacity) % this.capacity;
        this.size = this.size - 1;
        return item;
    }

    void print() {
        if (isEmpty()) {
            System.out.println("Deque is empty");
            return;
        }
        for (int i = 0; i < size; i++) {
            int index = (front + i) % capacity;
            System.out.print(array[index] + " ");
        }
        System.out.println();
    }
}

// This Class Use Switch Statment from Inputing User Input
class PrintAllClasses {
    public void MainMenu() {
        int userInput;
```

```
614         int RunAgainCode;
615         Scanner scanner = new Scanner(System.in);
616         System.out.println("\nDSA Assignment 1 \t Rida Ashraf - 070 - SE-22-A\n");
617         do {
618             System.out.println("1. Singly LinkedList");
619             System.out.println("2. Doubly LinkedList");
620             System.out.println("3. Circular LinkedList");
621             System.out.println("4. Stack using Array");
622             System.out.println("5. Stack using LinkedList");
623             System.out.println("6. Queue using Array");
624             System.out.println("7. Queue using LinkedList");
625             System.out.println("8. Circular Queue using Array");
626             System.out.println("9. Dequeue using Array");
627             System.out.print("Choose an Option: ");
628 
629             userInput = scanner.nextInt();
630             System.out.println("\n");
631 
632         switch (userInput) {
633             case 1: // Singly Linked List
634                 LinkedList linkedList = new LinkedList();
635                 System.out.println("Singly Linked List (Insertion Start): ");
636                 linkedList.insertAtStart(1);
637                 linkedList.insertAtStart(2);
638                 linkedList.print();
639                 System.out.println("Singly Linked List (Insertion End): ");
640                 linkedList.insertAtEnd(3);
641                 linkedList.insertAtEnd(4);
642                 linkedList.print();
643                 // Random insertion
644                 System.out.println("Singly Linked List (InsertionRandom): ");
645                 linkedList.insertRandom(2, 2);
646                 linkedList.print();
647                 System.out.println("Singly Linked List (Insertion Start Again): ");
648                 linkedList.insertAtStart(0);
649                 linkedList.insertAtStart(6);
650                 linkedList.print();
651                 System.out.println("\nSingly linked list Searching: ");
652                 System.out.println("The search Number is on index: " + linkedList.search(3));
653                 System.out.println("\nSingly Linked List (Deletion First): ");
654                 linkedList.deleteAtFirst();
655                 linkedList.print();
656                 System.out.println("Singly Linked List (Deletion End): ");
657                 linkedList.deleteAtEnd();
658                 linkedList.print();
659                 // Random deletion
660                 System.out.println("Singly Linked List (Deletion Random): ");
661                 linkedList.deleteRandom(1);
662                 linkedList.print();
663                 System.out.println("\n");
664                 break;
665             case 2:
666                 // Doubly Linked List
667                 DoublyLinkedList doublyList = new DoublyLinkedList();
668                 System.out.println("Doubly Linked List Insertion & Print: ");
669                 doublyList.insert(2);
```

```
670                 doublyList.insert(11);
671                 doublyList.insert(6);
672                 doublyList.insert(18);
673                 doublyList.insert(10);
674                 doublyList.print();
675                 System.out.println("\n");
676                 break;
677             case 3:
678                 // Circular Linked List
679                 CircularLinkedList circularList = new CircularLinkedList();
680                 System.out.println("Circular Linked List Insertion & Print: ");
681                 circularList.insert(1);
682                 circularList.insert(3);
683                 circularList.insert(5);
684                 circularList.insert(7);
685                 circularList.insert(9);
686                 circularList.print();
687                 System.out.println("\n");
688                 break;
689             case 4:
690                 // Stack using Array
691                 StackArray stackArray = new StackArray(10);
692                 System.out.println("Stack Using Array (Push, Pop, isFull , isEmpty, Peek,
    Print )");
693                 System.out.println("\nPush: ");
694                 stackArray.push(1);
695                 stackArray.push(3);
696                 stackArray.push(5);
697                 stackArray.push(7);
698                 stackArray.push(9);
699                 stackArray.print();
700                 System.out.println("Pop: ");
701                 stackArray.pop();
702                 stackArray.print();
703                 System.out.println("\nThe Peek is: " + stackArray.peek());
704                 System.out.println("The isEmpty and isFull Funtcions runs when the Stack is
    Empty or Full");
705                 System.out.println("\n");
706                 break;
707             case 5:
708                 // Stack using Linked List
709                 StackLinkedList stackLinkedList = new StackLinkedList();
710                 System.out.println("Stack Using Linked List (Push, Pop, isFull , isEmpty,
    Peek, Print )");
711                 System.out.println("\nPush: ");
712                 stackLinkedList.push(2);
713                 stackLinkedList.push(4);
714                 stackLinkedList.push(6);
715                 stackLinkedList.push(8);
716                 stackLinkedList.push(10);
717                 stackLinkedList.print();
718                 System.out.println("Pop: ");
719                 stackLinkedList.pop();
720                 stackLinkedList.print();
721                 System.out.println("\nThe Peek is: " + stackLinkedList.peek());
722                 System.out.println("The isEmpty and isFull Funtcions runs when the Stack is
    Empty or Full");
```

```
723                 System.out.println("\n");
724                 break;
725             case 6:
726                 // Queue using Array
727 System.out.println("Queue using Array ( Enqueue, Dequeue, isEmpty, isFull,getFront,
    getRear, Print)");
728                 QueueArray queueArray = new QueueArray(10);
729                 System.out.println("\nEnqueue: ");
730                 queueArray.enqueue(10);
731                 queueArray.enqueue(20);
732                 queueArray.enqueue(30);
733                 queueArray.enqueue(40);
734                 queueArray.enqueue(50);
735                 queueArray.print();
736                 System.out.println("Dequeue: ");
737                 queueArray.dequeue();
738                 queueArray.print();
739                 System.out.println("\nThe Front is: " + queueArray.getFront());
740                 System.out.println("The Rear is: " + queueArray.getRear());
741 System.out.println("The isEmpty and isFull Funtcions runs when the Queue isEmpty or Full");
742                 System.out.println("\n");
743                 break;
744             case 7:
745                 // Queue using Linked List
746                 System.out.println(
747                         "Queue using Linked List( Enqueue, Dequeue, isEmpty, isFull,
    getFront, getRear, Print)");
748                 QueueLinkedList queueLinkedList = new QueueLinkedList();
749                 System.out.println("\nEnqueue: ");
750                 queueLinkedList.enqueue(50);
751                 queueLinkedList.enqueue(40);
752                 queueLinkedList.enqueue(30);
753                 queueLinkedList.enqueue(20);
754                 queueLinkedList.enqueue(10);
755                 queueLinkedList.print();
756                 System.out.println("Dequeue: ");
757                 queueLinkedList.dequeue();
758                 queueLinkedList.print();
759                 System.out.println("\nThe Front is: " + queueLinkedList.getFront());
760                 System.out.println("The Rear is: " + queueLinkedList.getRear());
761                 System.out.println("The isEmpty and isFull Functions runs when the Queue is
    Empty or Full");
762                 System.out.println("\n");
763                 break;
764             case 8:
765                 // Circular queue Using Array
766                 CircularQueue circularQueue = new CircularQueue(10);
767                 System.out.println("Circular Queue (Enqueue):");
768                 circularQueue.enqueue(1);
769                 circularQueue.enqueue(2);
770                 circularQueue.enqueue(3);
771                 circularQueue.enqueue(4);
772                 circularQueue.enqueue(5);
773                 circularQueue.print();
774
775                 System.out.println("Circular Queue (Dequeue):");
```

```
                circularQueue.dequeue();
                circularQueue.dequeue();
                circularQueue.print();

                System.out.println("\nThe Front is: " + circularQueue.getFront());
                System.out.println("The Rear is: " + circularQueue.getRear());
                System.out.println("The isEmpty and isFull Funtcions runs when the Circular Queue is Empty or Full");
                System.out.println("\n");
                break;
            case 9:
                // Double Ended Queue (Deque) Using Array
                Deque deque = new Deque(10);
                System.out.println("Double Ended Queue (Enque Rear): ");
                deque.enqueueRear(1);
                deque.enqueueRear(2);
                deque.enqueueRear(3);
                deque.enqueueRear(4);
                deque.enqueueRear(5);
                deque.print();
                System.out.println("Double Ended Queue (Enque Front): ");
                deque.enqueueFront(4);
                deque.enqueueFront(19);
                deque.print();

                System.out.println("\nDouble Ended Queue (Dequeue Front):");
                deque.dequeueFront();
                deque.print();
                System.out.println("Double Ended Queue (Dequeue Rear):");
                deque.dequeueRear();
                deque.print();

                System.out.println("\nThe Front is: " + deque.dequeueFront());
                System.out.println("The Rear is: " + deque.dequeueRear());
                System.out.println("\n");
                break;
            default:
                System.out.println("Invalid Choice……. ");
                break;

            }
        System.out.println("\nHello if you want to run again this program then Press 1……. ");
        RunAgainCode = scanner.nextInt();

        } while (RunAgainCode == 1);
        scanner.close();
    }
}

public class Main {
    public static void main(String[] args) {
        PrintAllClasses MainMenu = new PrintAllClasses();
        MainMenu.MainMenu();
    }
}
```

Outputs:

```
DSA Assignment 1         Rida Ashraf - 070 - SE-22-A

1. Singly LinkedList
2. Doubly LinkedList
3. Circular LinkedList
4. Stack using Array
5. Stack using LinkedList
6. Queue using Array
7. Queue using LinkedList
8. Circular Queue using Array
9. Dequeue using Array
Choose an Option:
```

```
Singly Linked List (Insertion Start):
2 -> 1 -> Null
Singly Linked List (Insertion End):
2 -> 1 -> 3 -> 4 -> Null
Singly Linked List (InsertionRandom):
2 -> 1 -> 2 -> 3 -> 4 -> Null
Singly Linked List (Insertion Start Again):
6 -> 0 -> 2 -> 1 -> 2 -> 3 -> 4 -> Null

Singly linked list Searching:
The search Number is on index: 5

Singly Linked List (Deletion First):
0 -> 2 -> 1 -> 2 -> 3 -> 4 -> Null
Singly Linked List (Deletion End):
0 -> 2 -> 1 -> 2 -> 3 -> Null
Singly Linked List (Deletion Random):
0 -> 1 -> 2 -> 3 -> Null



Hello if you want to run again this program then Press 1....
```

```
Doubly Linked List Insertion & Print:
10 -> 18 -> 6 -> 11 -> 2 -> Null



Hello if you want to run again this program then Press 1....
```

```

Circular Linked List Insertion & Print:
1 3 5 7 9



Hello if you want to run again this program then Press 1....
```

```
Stack Using Array (Push, Pop, isFull , isEmpty, Peek, Print )

Push:
1 3 5 7 9
Pop:
1 3 5 7

The Peek is: 7
The isEmpty and isFull Funtcions runs when the Stack is Empty or Full



Hello if you want to run again this program then Press 1....
```

```
Stack Using Linked List (Push, Pop, isFull , isEmpty, Peek, Print )

Push:
10 8 6 4 2
Pop:
8 6 4 2

The Peek is: 8
The isEmpty and isFull Funtcions runs when the Stack is Empty or Full



Hello if you want to run again this program then Press 1....
```

```
Queue using Array ( Enqueue, Dequeue, isEmpty, isFull, getFront, getRear, Print)

Enqueue:
10 20 30 40 50
Dequeue:
20 30 40 50

The Front is: 20
The Rear is: 50
The isEmpty and isFull Funtcions runs when the Queue is Empty or Full



Hello if you want to run again this program then Press 1....
```

```
Queue using Linked List( Enqueue, Dequeue, isEmpty, isFull, getFront, getRear, Print)

Enqueue:
50 40 30 20 10
Dequeue:
40 30 20 10

The Front is: 40
The Rear is: 10
The isEmpty and isFull Functions runs when the Queue is Empty or Full



Hello if you want to run again this program then Press 1....
```

```
Circular Queue (Enqueue):
1 2 3 4 5
Circular Queue (Dequeue):
3 4 5

The Front is: 3
The Rear is: 5
The isEmpty and isFull Funtcions runs when the Circular Queue is Empty or Full



Hello if you want to run again this program then Press 1....
```

```
Double Ended Queue (Enque Rear):
1 2 3 4 5
Double Ended Queue (Enque Front):
19 4 1 2 3 4 5

Double Ended Queue (Dequeue Front):
4 1 2 3 4 5
Double Ended Queue (Dequeue Rear):
4 1 2 3 4

The Front is: 4
The Rear is: 4



Hello if you want to run again this program then Press 1....
```