IARE

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

LABORATORY WORK BOOK

Name of the Student : Racherla Santhosh

Class IT-B Semester 03

Course Code : ACSD11 Course Name : DS laboratory

Name of the Course Faculty Ms. K. Laxminarayanamma Faculty ID : IARE 10033

Exercise Number : 06 Week Number : 06 Date : 15/10/2024

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

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED					
			Aim/ Preparation	Algorithm / Procedure Performance in the Lab	Source Code Calculations and Graphs	Program Execution Results and Error Analysis	Viva - Voce	Total
			4	4	4	4	4	20
1	6.1	Linear Queue						
2	6.2	Stack Using Queues						
3	6.3	Queue Using Stacks						
4	6.4	Circular Queue						
5	6.5	Deque	4	4	4	4	4	20
6								
7								
8								
9								
10								
11								
12								

Santhosh

Signature of the Student

Signature of the Faculty

6. Queue.

6.1 Linear Queue :-

AIM :- Write a Program using Linear queue as its function.

PROGRAM :-

```
import java.util.Scanner;
Public class LinearQueue {
    Private Static final int MAX = 5;
    Private Static int[] queue = new int[MAX];
    Private Static int front = -1;
    private Static int Year = -1;
    Public Static boolean isEmpty() {
        Yeturn front == -1;
    }

    Public Static boolean isFull() {
        Yeturn Year == MAX - 1;
    }
```

```
Public static void enqueue (int item) {
  if (isFull()) {
    System.out.println("queue is full. Cannot
                        enqueue.");
  } else {
    if (front == -1) {
      front = 0;
    }
    rear ++;
    queue [rear] = item;
    System.out.println (item + " enqued to the
                        queue.");
  }
}

Public static void dequeue () {
  if (isEmpty()) {
    System.out.println ("Queue is empty. Cannot
                        dequeue.");
  } else {
    System.out.println (queue [front] + "
        dequeued from the queue.");
        front ++ ;
        if (front > Year) {
            front = rear = -1;
        }
    }
}

Public Static void display() {
    if (isEmpty()) {
        System.out.println (" queue is empty");
    } else {
        System.out.print (" queue elements :");
        for (int i = front ; i <= Year ; i++) {
            System.out.print (queue [i] + "  ");
        }
        System.out.println() ;
    }
}

Public Static void main ( String[] args ) {
    Scanner scanner = new Scanner (System.in) ;
```

```
int choice, item;
do {
    System.out.println("\n1. Enqueue");
    System.out.println("2. Dequeue");
    System.out.println("3. Display");
    System.out.println("4. Exit");
    System.out.print("Enter your choice:");
    choice = Scanner.nextInt();
    Switch (choice) {
      case 1:
        System.out.print("Enter item to enqueue:");
        item = Scanner.nextInt();
        enqueue(item);
        break;
      case 2:
        dequeue();
        break;
      case 3:
        display();
        break;
        case 4 :
          System.out.println (" Exiting .....");
          break;
        default :
          System.out.println (" Invalid choice, please
                                       Try Again");
      }
  } while (choice != 4) ;
     Scanner.close() ;
  }
}
```

RESULT :-

INPUT : 1) Enqueue

2) Dequeue

3) Display

4) Exit

Enter Your choice : 1

Enter item of enqueue : 20

OUTPUT : The Program is executed Successfully.

20 enqued to the queue.

Enter Your choice : 28

Queue element : 20.

6.2 Stack Using Queues :-

AIM :- Write a Program On Implementing a LIFO Stack using only two queues. The implemented Stack should Support all the functions of a normal Stack (Push, top, pop & empty).

PROGRAM :-

```
import java.util.LinkedList;
import java.util.queue;
class MyStack {
  Private Queue < Integer > queue1;
  Private Queue < Integer > queue2;
  Public MyStack () {
    queue1 = new LinkedList <>();
    queue2 = new LinkedList <>();
  }
  Public void Push (int x) {
    queue2.add(x);
    while (!queue1.isEmpty()) {
      queue 2.add (queue 1. remove());
    }
    Queue < Integer > temp = queue 1;
    queue 1 = queue 2;
    queue 2 = temp;
  }

  Public int pop() {
    if (! queue 1. isEmpty()) {
      return queue 1. remove();
    }
    return -1;
  }

  Public int top() {
    if (! queue 1. isEmpty()) {
      return queue 1. Peek();
    }
    return -1;
  }

  Public boolean empty() {
        return queue1.isEmpty();
    }
    Public static void main(String[] args) {
      My Stack stack = new MyStack();
       stack.push(1);
       stack.push(2);
      System.out.println("Top element:" + stack.top());
      System.out.println("popped element:" + stack.pop());
     System.out.println("Is stack empty ?" +
                                 stack.empty());
    }
}
```

RESULT :-

INPUT : ["MyStack", "Push", "push", "top", "pop", "empty"]

[[], [1], [2], [], [], []]

OUTPUT : [null, null, null, 2, 2, false]

Top Element : 1

Popped Element : 2

Is stack empty : False

6.3 Queue Using Stacks :-

AIM :- Write a Program On Implementing a FIFO queue using only two stacks. The implemented queue should support all the functions of a normal queue (Push, Peek, pop, and empty).

PROGRAM :-

```
import java.util. Stack;
class Myqueue {
  Private Stack < Integer > stack1;
  Private Stack < Integer > stack2;
  Public Myqueue() {
    stack1 = new Stack <>();
    stack2 = new Stack <>();
  }

  Public void Push (int x) {
     stack1. push (x);
  }
```

```
    Public int pop() {
      if (Stack2.isEmpty()) {
        while (!Stack1.isEmply()) {
          Stack2.push(Stack1.pop());
        }
      }
      return Stack2.pop();
  }
  Public int Peek() {
    if (Stack2.isEmpty()) {
     while (!Stack1.isEmpty()) {
       Stack2.push (Stack1.pop());
     }
  }
  return Stack2.peek();
}
Public boolean empty() {
   Return Stack1.isEmpty() && Stack2.isEmpty();
}
Public Static void main(String[] args) {
        Myqueue queue = new Myqueue();
          queue.push(1);
          queue.push(2);
         System.out.println("Front element:" + queue.peek());
         System.out.println("Popped element:" + queue.pop());
         System.out.println("Is queue empty ?" + queue.empty());
       }
    }
```

RESULT :-

```
INPUT : ["Myqueue", "push", "push", "peek", "pop", "empty"]
        [[], [1], [2], [], [], []]
OUTPUT : [null, null, null, 1, 1, false]
Front Element : 1
Popped Element : 1
Is Queue Empty : False
```

6.4 Circular Queue :-

AIM :- Write a Program based on function the Circular queue.

PROGRAM :-

```
class Circularqueue {
   Private int size;
   Private int front, rear;
   Private int [] queue ;
   Public CircularQueue (int size) {
      this. size = size;
      this. queue = new int [size];
      this. front = this. rear = -1;
   }
  Public void enqueue (int data) {
     if ((rear + 1) % size == front) {
      System. out. println (" queue is full. Cannot
                                  enqueue" + data);
     } else {
        if ( front == -1) {
            front = 0;
        }
        rear = ( rear + 1) % size ;
        queue [rear] = data ;
        System.out.println ( data + " enqueued to the queue");
    }
}
Public int dequeue () {
    if ( front == -1) {
     System.out.println ( "queue is empty. Cannot
                                   dequeue");
     return -1;
    }
    int dequeuedElement = queue [front] ;
   System.out.println (dequeued Element + " dequeued
                             from the queue");
      if ( front == rear) {
         front = rear = -1 ;
        } else {
            front = ( front + 1 ) % size ;
        }
      return dequeuedElement ;
   }
   Public int getFront() {
      if ( front = = -1)   {
         System. out. println (" Queue is empty") ;
         return -1 ;
      }
      return queue [front] ;
  }
  Public int getRear () {
     if ( rear = = -1) {
        System. out. println (" Queue is empty") ;
        return -1 ;
     }
     return queue[ rear] ;
  }
```

```
Public boolean isEmpty() {
  return front == -1;
}
Public boolean isFull() {
  return (rear + 1) % size == front;
}
Public void display() {
  if (front == -1) {
    System.out.println("queue is empty");
  } else {
    System.out.print(" queue elements:");
    int i = front;
    while (i != rear) {
      System.out.print(queue[i] + " ");
      i = (i+1) % size;
    }
   System.out.print(queue[rear] + "\n");
  }
}
```

RESULT :-

```
OUTPUT :-  10  enqued  to  the  queue      Front Element : 30
           20    "      "    "    "        Rear Element : 40
           30    "      "    "    "        Display : 30  40
           40    "      "    "    "
           50    "      "    "    " .
```

6.5 Deque (Doubly Ended Queue) :-

AIM :- Write a Program using types of Dequeue.

PROGRAM :-

```
import java.util.ArrayDequeue;
import java.util.Deque;
import java.util.Iterator;
Public class DequeOperations {
  Public static void main ( String [] args ) {
    Deque < Integer > deque = new ArrayDeque <>() ;
    deque.addlast (10);
    deque.addLast (20);
    deque.addlast (30);
    System.out.println ("Deque after addlast (append): "+ deque);
    deque.addFirst(0);
    System.out.println("Deque after addFirst (appendLeft): "+ deque);
    int PoppedRight = deque.removeLast();
    System.out.println("Element popped from right: "+ PoppedRight);
    System.out.println("Deque after removeLast (pop): "+ deque);
    int poppedleft = deque.removeFirst();
    System.out.println("Element popped from left: "+ Poppedleft);
    System.out.println("Deque after removeFirst (popleft): "+ deque);
  }
Public static int getIndex(Deque<Integer> deque, int ele) {
   int index = 0;
   for (int value : deque) {
     if (value == ele) {
        return index;
     }
     index++;
  }
  return -1;
}
```

```
Public Static void rotateRight ( Deque < Integer > deque , int Steps) {
  for (int i = 0; i < Steps; i++) {
    deque . add First ( deque . removeLast () ) ;
  }
}
Public Static void rotateLeft ( Deque < Integer > deque , int steps) {
  for ( int i = 0; i < Steps ; i++ ) {
    deque . addlast ( deque . remove First () ) ;
  }
}
}
```

RESULT :-

```
OUTPUT : Deque after addlast (append) : [10, 20, 30]
         Deque after add First (append left) : [0, 10, 20, 30]
         Element popped from right : 30
         Deque after removeLast (Pop) : [0, 10, 20]
         Element Popped from left : 0
         Deque after remove First (popleft) : [10, 20]
         Index of element 20 : 1
         Deque after inserting 15 at index 1 : [10, 15, 20]
         "     "  removing first occurence of 15 : [10, 20]
```

VIVA VOCE :-

1) What is queue ?

A) A queue is a Data Structure that follows the FIFO Principle, where the first item added is the first to be removed.

2) what is the difference b/w Stack using queues and queue using Stacks ?

A) Stack Using queues : Uses two queues to Simulate LIFO.

queue Using Stacks : Uses two Stacks to Simulate FIFO.

3) What is Circular queue ?

A) It is an extended Version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. The Operations are Performed based on FIFO principle, It is also called "Ring Buffer".

4) What is Deque ?

A) A Deque is a Double - ended queue that allows insertion & removal of elements from both front & back.