G.L.A UNIVERSITY

ऋते ज्ञानान्न मुक्तिः

SESSION :- 2020-2021

ASSIGNMENT FILE

TOPIC:- DATA STRUCTURES AND ALIGRITHMS

ASSIGNMENT Ⅳ

COURSE :- B.TECH (CSE)

SECTION :- P

ROLL NO :- 13 (191500201)

Submitted to:

MR. Sharad Gupta

Submitted by:

Ayush Sharma

11 What is the difference between Peek and Poll ?

Peek() :- This Method returns the object at the top of the current queue, without removing it. If the queue is empty this Method returns null.

Parameters :- The method does not take any parameters.

Exception :- The Method throws EmptyStackException if stack is empty.

Ex :-

```
import java.util.Iterator;
import java.util.LinkedList;
import java.Queue;
Public class QueueExample
{
    Public static void main(String args[])
    {
        Queue<String> queue = new LinkedList<String>();
        queue.add("Java");
        queue.add("Java fx");
        System.out.Println("TOP Element :"+queue.Peek());
        Iterator<String>it = queue.iterator();
        System.out.Println("contents of queue");
        While(it.hasNext())
        {
            System.out.Println(it.next());
        }
    }
}
```

Output :-

Top Element :
contents of queue

Poll() :- This Method is used to retrive and remove the head of this queue. If queue is empty then it will returns null. The Method does not throws an exception when Queue is Empty.

Returns :- This Method returns the elements at the front of the container or the head of the queue. It returns null when Queue is empty.

Ex :-

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Queue;
Public class QueueExample
 {
    Public static void main(String args[])
     {
       Queue<String> queue = new LinkedList<String>();
       queue.add("Java");
       queue.add("Java fx");
       System.out.Println("Element at the top of the queue:" + queue.Poll());
       Iterator<String> it = queue.iterator();
        System.out.Println("contents of queue");
        while(it.hasNext())
          {
            System.out.Println(it.hasNext());
          }
      }
}
```

Output :-

iii) What is the difference between Peek and POP?

Peek():- The Peek() Method returns the element of the top of the stack but does not remove it.

POP():- The POP() Method removes and returns the top element of the stack.

Note:- The term POP are usually used for stack, not queue or LinkedList.

Ex:-

```
import java.util.*;
class Myclass
{
  Public static void main (String args[])
   {
     Stack <integer> even = new stack < > ();
     even.Push(0);
     even.Push(2);
     even.Push(4);
     even.Push(6);
    System.out.println("Print stack beyore POP:");
    System.out.println(even);
    System.out.println("POP=>"+even.Pop());
    System.out.println("Print stack after POP:");
    System.out.println(even);
    System.out.println("No. on top of the stack is:"+ even.Peek());
}
}
```

iii : Write a program of stack to insert the data with even numbers only by using array and linked list.(Implement only push function).

```
import java.util.Scanner;
class EvenNode {
   int data;
   EvenNode next;
}
public class EvenStackArr {
   EvenNode top;
   EvenStackArr() {
      top=null;
   }
   public static boolean checkEven(int n) {
      if (n%2==0) {
         return  true;
      }
      return false;
   }
   void push() {
System.out.println("Please! Enter Data..");
Scanner sc = new Scanner(System.in);
int number = sc.nextInt();
if (checkEven(number)==true) {
   EvenNode node = new EvenNode();
   node.data=number;
   node.next=top;
   top=node;
   System.out.println("Data Inserted...");
}
else {
   System.out.println(number + " is not a valid entry!!!");
}
   }
   public static void main(String[] args) {
      EvenStackArr obj = new EvenStackArr();
      while (true) {
         obj.push();
      }
   }
}
```

iv : Write a program of singly linked list. Insert the data with prime numbers only.

```
import java.util.*;
class PrimeNode
{
   int primeNumber;
   PrimeNode next;
}
public class PrimeLinkedList
{
   PrimeNode head;
   PrimeLinkedList()
   {
      head=null;
   }
   public static int checkPrimeNumber(int number)
   {
      int counter=0;
      for (int i=2;i<=number/2;i++)
      {
         if (number%i==0)
         {
            counter=1;
            break;
         }
      }
      if (counter==0) {
         return number;
      }
      return number/2;
   }
   void addPrimeNode()
   {
      System.out.println("Enter a prime number...");
      Scanner sc2 = new Scanner(System.in);
      int n = sc2.nextInt();
      if (checkPrimeNumber(n)==n){
         PrimeNode newnode = new PrimeNode();
```

```
        newnode.primeNumber = n;
        newnode.next = null;
        if (head==null){
           head=newnode;
        }
        else {
           PrimeNode current = head;
           while (current.next!=null){
              current=current.next;
           } current.next=newnode;
        }
        System.out.println("A prime number inserted...");
     }
     else {
        System.out.println("Try Again!!!");
     }
  }
  void traverseLL()
  {
     if (head==null) {
        System.out.println("LinkedList is empty...");
     }
     else {
        PrimeNode current;
        for (current=head;current!=null;current=current.next) {
           System.out.println(" " + current.primeNumber);
        }
     }
  }
  public static void main(String[] args) {
     PrimeLinkedList obj = new PrimeLinkedList();
     while (true) {
        System.out.println("Press 1 to insert a prime number!");
        System.out.println("Press 2 to traverse the Linkedlist!");
        Scanner sc = new Scanner(System.in);
        int choice=sc.nextInt();
        if (choice==1) obj.addPrimeNode();
        else if (choice==2) obj.traverseLL();
        else System.exit(0);   }
  }        }
```

V : Program - You have an array containing the roll numbers of the student. Import this data into singly linked list.

```
class Node
{
   int data;
   Node next;
}
public class Array2LinkedList
{
   static Node insert(Node root, int item)
   {
      Node temp = new Node();
      Node ptr;
      temp.data = item;
      temp.next = null;

      if (root == null)
         root = temp;
      else
      {
         ptr = root;
         while (ptr.next != null)
            ptr = ptr.next;
         ptr.next = temp;
      }
      return root;
   }
   static void display(Node root)
   {
      while (root != null)
      {
         System.out.print( root.data + " ");
         root = root.next;
      }
   }
   static Node arrayToList(int arr[], int n)
   {
      Node root = null;
```

```
    for (int i = 0; i < n; i++)
        root = insert(root, arr[i]);
    return root;
}
public static void main(String args[])
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int n = arr.length;
    Node root = arrayToList(arr, n);
    display(root);
  }
}
```

VII] What are infix and Postfix notations (Polish notations)?

The way to write arithmetic expression is known as a notation. An arithmetic expression can be written in three different but equivalent notations, i.e,

Infix Notation :- We write expression in infix notation, e.g, a-b+c, where operators are used in-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorith to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix Notation :- It is also called Polish notation. In this notation, operator is prefixed to operands, i.e. operator is written ahead of operands. for example, +ab. This is equivalent to its infix notation a+b.

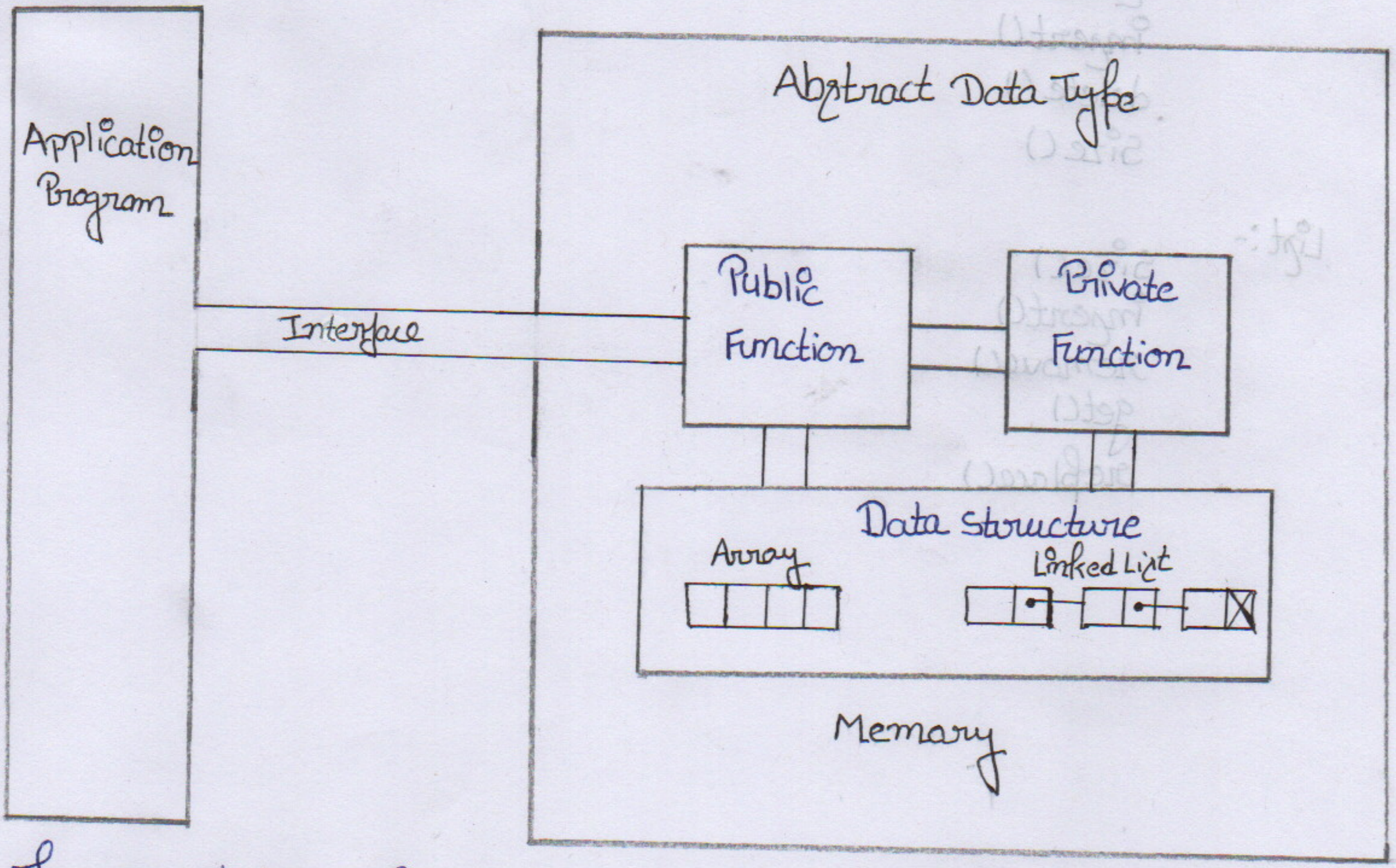
Postfix Notation :- This notation style is known as Reversed Polish Notation. In this notation style, the operator is Postfixed to the operands i.e, the operator is written after the operands. for example, ab+. this equivalent to its infix notation a+b.

The difference in all three notations :-

Sr.No	Infix	Prefix	Postfix
1	a+b	+ab	ab+
2	(a+b)*c	*+abc	ab+c*
3	a*(b+c)	*a+bc	abc+*
4	a/b+c/d	+/ab/cd	ab/cd/+
5	(a+b)*(c+d)	*+ab+cd	ab+cd+*
6	((a+b)*c)-d	-*+ab cd	ab+c* d-

ix) What are abstract data types?

Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations. The define of ADT only Mentions what operations are to be Performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. it is called "abstract" because it gives an implementation-independent view. The Process of Providing only the essentials and hiding the details is known as abstraction.



The user of data type does not need to know How that data type implemented, for example we have been using Primitive values like, int, float, char data type can operate and performed.

So a user only needs to know what a data type can do, but not how it will be implemented. Think of ADT as a black box which hides the inner structure and design of the data type.

Some examples of ADT are Stack, Queue, List etc.
Let us see some operations of those mentioned ADT -

Stack :-

- isFull()
- Push()
- POP()
- Peek()
- size()

Queue :-

- isFull()
- insert()
- delete()
- size()

List :-

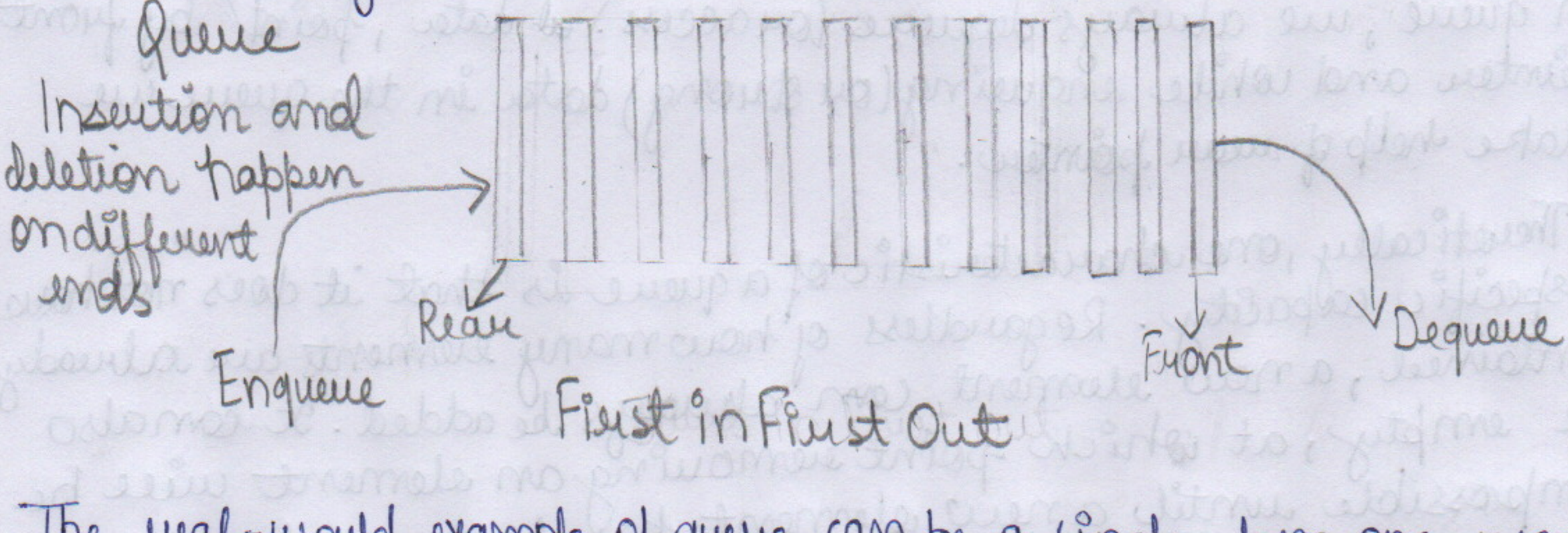
- size()
- insert()
- remove()
- get()
- replace()

X| What are Big-O notations? Describe the increasing order and decreasing order.

Big O notation is one of the most fundamental tools for Programmers to analyze the time and space complexity of an algorithms. Big O notation is an asymptotic notation to measure the upper bound performance of an algorithm. your choice of algorithm and data structure Matters when you write software with strict SLAs or large Programs. Big O notation allows you to compare algorithm performance to find the best for your given situation. the Big O rating helps to ensure you have a lower running time than competitors.

xi] What is a queue?

Queue is an abstract data structure, similar to Stacks, Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good exampl of a queue is any queue of consumers for a resource where the consumer that come first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



The real-would example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-would examples can be seen as queues at the ticket windows and bus stops.

Basic Operations

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues-

- enqueue() - add (store) an item to the queue.
- dequeue() - remove (access) an item from the queue.

* Front : Get the front item from queue

* Rear : Get the last item from queue.

Few more functions are required to make the above-mentioned queue operation efficent. These are -

- peek() - Gets the element at the front of the queue without removing it.
- isfull() - Checks if the queue is full
- isempty() - Checks if the queue is empty.

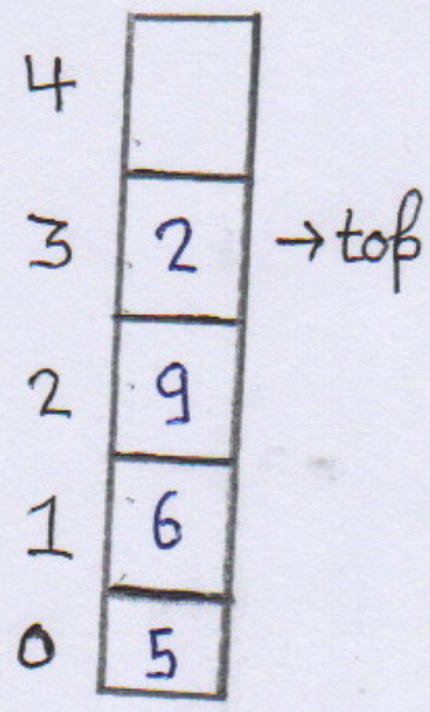
In queue, we always dequeue (or access) data, point by front pointer and while enquing (or storing) data in the queue we take help of rear pointer.

Theoretically, one characteristic of a queue is that it does not have a specific capacity. Regardless of how many elements are already contained, a new element can always be added. It can also be empty, at which point removing an element will be impossible until a new element has been added again.

XII] What are the difference between Stack and queue?

Stack :- A Stack is a linear data structure in which elements can be inserted and deleted only form one side of the List called the top. A Stack follows the LIFO (Last in first out) Principal, ie, the element inserted at the Last is the first element to come out. the insertion of an elements into stack is called Push operation, and deletion of an element from the stack is called POP operation. In stack we always keep track of the last element present in the List with a pointer called top.

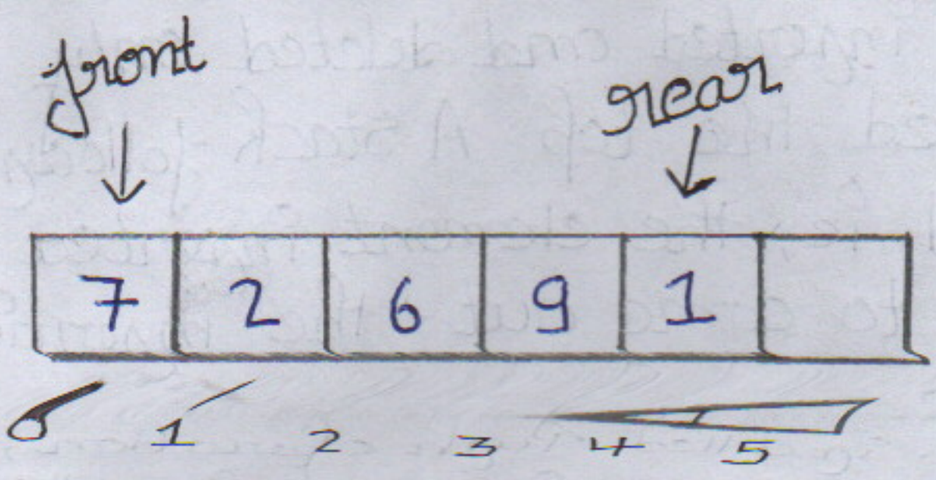
The diagrammatic representation of stack is given below:-



Stack

Queue :- A queue is a linear data structure in which elements can be inserted only from one side of the List called rear, and the elements can be deleted only from the other side called the front. the queue data structure follows the FIFO (first in first out) Principle, i.e. the elements Inserted at first in the List, is the first element to be removed from the List. The insertion of an element in a queue is called an enqueue operation and the deletion of an element is called a dequeue Operation. in queue we always maintain two Pointers, one Pointing to the element which was inserted at the first and still Present in the List with the front Pointer and second Pointer Pointing to the element inserted at the Last with the rear Pointer.

The digrammatic representation of queue is given below:-



Queue

XIII: Write a program of singly queue using array.

```
import java.util.Scanner;
public class SinglyQueue
{
   int ar[];
   int Front , Rear;

   SinglyQueue()
   {
      ar = new int[5];
      Front = -1;
      Rear = -1;
   }
   void insert()
   {
      if (Rear==4)
      {
         System.out.println("Queue full");
      }
      else
      {
         System.out.println("Please! Enter Data...");
         Scanner sc2 = new Scanner(System.in);
         int data=sc2.nextInt();

         if (Front==-1)
         {
            Front = 0;
         }
         Rear = Rear+1 ;
         ar[Rear] = data;
         System.out.println("DATA INSERTED....");
      }
   }
   void delete()
   {
      if (Front==-1 || Rear==-1)
      {
         System.out.println("Queue Empty");
      }
      else
      {
```

```
if (Front==Rear)
{
   System.out.println("Delete :- " + ar[Front]);
   Front= -1;
   Rear= -1;
}
else
{
   System.out.println("Delete :- " + ar[Front]);
      Front = Front + 1;
    }
  }
}
void traverse()
{
  if (Front==-1 || Rear==-1)
  {
    System.out.println("Queue Empty");
  }
  else
  {
    System.out.println("-------Queue Elements-------");
    for (int i = Front; i<=Rear; i++)
    {
      System.out.println(" " + ar[i]);
    }
  }
}
void peek()
{
  if (Front==-1 || Rear==-1)
  {
    System.out.println("Queue empty");
  }
  else
  {
    System.out.println("Last element :- " + ar[Rear]);
  }
}
void poll()
{
  if (Front==-1 || Rear==-1)
  {
```

```
            System.out.println("List Empty");
        }
        else
        {
            System.out.println("First element :- " + ar[Front]);
        }
    }
    public static void main(String args[])
    {
        SinglyQueue ob = new SinglyQueue();

        while (true)
        {
            System.out.println("PRESS 1 FOR INSERT");
            System.out.println("PRESS 2 FOR DELETE");
            System.out.println("PRESS 3 FOR TRAVERSE");
            System.out.println("PRESS 4 FOR PEEK");
            System.out.println("PRESS 5 FOR POLL");
            System.out.println("PRESS 6 FOR EXIT");

            System.out.println("PLEASE! ENTER YOUR CHOICE");
            Scanner scanner = new Scanner(System.in);
            int ch = scanner.nextInt();

            switch (ch)
            {
                case 1:
                    ob.insert();
                    break;
                case 2:
                    ob.delete();
                    break;
                case 3:
                    ob.traverse();
                    break;
                case 4:
                    ob.peek();
                    break;
                case 5:
                    ob.poll();
                    break;
                case 6:
                    System.exit(0);
```

```
            break;
        default:
            System.out.println("invalid choice");
    }
  }
 }
}
```

XIV: Write a program of circular queue using array.

```
import java.util.Scanner;
public class CircularQueue
{
    int ar[];
   int Front , Rear;

   CircularQueue()
   {
      ar = new int[5];
      Front = -1;
      Rear = -1;
   }
   void insert()
   {
      if (Front==0 && Rear==4 || Rear==Front-1)
      {
         System.out.println("Circular queue is full");
      }
      else
      {
         System.out.println("Please! Enter Data");
         Scanner scanner2 = new Scanner(System.in);
         int data = scanner2.nextInt();
         if (Front==-1)
         {
            Front=0;
         }
         if (Rear<4)
         {
            Rear = Rear +1;
         }
         else if (Rear==4 && Front!=0)
         {
            Rear=0;
         }
         ar[Rear] = data;
         System.out.println("data inserted...");
```

```
    }
  }
  void delete()
  {
    if (Front==-1 || Rear==-1)
    {
      System.out.println("Circular empty");
    }
    else
    {
      if (Front==Rear)
      {
        System.out.println("Deleted :- " + ar[Front]);
        Front=-1;
        Rear=-1;
      }
      else  if (Front<Rear)
      {
        System.out.println("Deleted :- " + ar[Front]);
        Front=Front +1;
      }
      else if (Front==4)
      {
        System.out.println("Deleted :- " + ar[Front]);
        Front=0;
      }
      else if (Rear<Front)
      {
        System.out.println("Deleted :- " + ar[Front]);
        Front=Front+1;
      }
    }
  }
  void traverse()
  {
    if (Front==-1 || Rear==-1)
    {
      System.out.println("Circular queue empty");
    }
    else
```

```
{
    if (Front<=Rear)
    {
        for (int i = Front; i<=Rear; i++)
        {
            System.out.println(" " + ar[i] );
        }
    }
    else
    {
        for (int i=Front; i<=4;i++)
        {
            System.out.println(" "+ ar[i]);
        }
        for (int i=0; i<=Rear;i++)
        {
            System.out.println(" " + ar[i]);
        }
    }
}
public static void main(String args[])
{
    CircularQueue obj = new CircularQueue();

    while (true)
    {
        System.out.println("PRESS 1 FOR INSERT");
        System.out.println("PRESS 2 FOR DELETE");
        System.out.println("PRESS 3 FOR TRAVERSE");
        System.out.println("PRESS 4 FOR EXIT");
        System.out.println("PLEASE! ENTER YOUR CHOICE");

        Scanner scanner = new Scanner(System.in);
        int ch = scanner.nextInt();

        switch (ch)
        {
            case 1:
                obj.insert();
```

```
            break;
        case 2:
            obj.delete();
            break;
        case 3:
            obj.traverse();
            break;
        case 4:
            System.exit(0);
        default:
            System.out.println("invalid choice");
        }
    }
}
```

XV: Write a program of singly queue using linked list.

```
import java.util.Scanner;
class Node
{
   int data;
   Node next;
}
public class SinglyQueueLinkedList
{
   Node Front , Rear;
   SinglyQueueLinkedList()
   {
      Front=null;
      Rear=null;
   }
   void insert()
   {
      System.out.println("Please! Enter Data");
      Scanner scanner2 = new Scanner(System.in);
      int item = scanner2.nextInt();

      Node newnode = new Node();
      newnode.data=item;
      newnode.next=null;

      if (Front==null)
      {
         Front=newnode;
         Rear=newnode;
      }
      else
      {
         Node current = Rear;
         while (current.next!=null)
         {
            current=current.next;
         }
         current.next=newnode;
```