INTRO TO DSA - MODULE 2

Introduction (3rd February)

1. What are functions ?
2. Why do we use functions ? (modularity, easy debugging and reusability of code)
3. Return type, arguments and analysis of functions
4. Context associated with functions
5. Flow of execution of functions
6. Be clear that the moment a function starts running,its context is created and the context starts playing.When a function call is made, the caller context pauses at that line and new context for the called function starts running. Once the context of the called function is destroyed we come back to the paused location in the caller context and the caller context starts playing.
7. What is a stack ? (LIFO data structure)
8. What is call stack ?
9. What is recursion ?
10. Why do we study recursion ? (Divide and Conquer Technique)
11. Examples of divide and conquer in real life
12. Recursion as black magic as you just have to call smaller subproblems and believe them to give the answers and then form the answer to your current problem. But you also have to write the base case.
13. Parts of Recursion - Base Case, Relationship between bigger and smaller subproblem
14. Stack Overflow and importance of base case
15. Print Acciojob n times

Flow of execution of functions

```
public class Main
{
	public static void main(String[] args) {
		System.out.println("Main");
		f1(3);
		System.out.println("Done");
	}
	public static void f1(int a)
	{
	    System.out.println(a);
	    System.out.println("Inside f1");
```

```
        f2();
        System.out.println("yo");
    }
    public static void f2()
    {
        int a = f3();
        System.out.println(a);
        System.out.println("Inside f2");
        return;
    }
    public static int f3()
    {
        f4();
        return 5;
    }
    public static void f4()
    {
        return;
    }
}
```

Print Acciojob n times

```
import java.io.*;
import java.util.*;

class Main
{
    public static void main(String args[])throws IOException
    {
        Scanner sc = new Scanner(System.in);

        int n=sc.nextInt();

        solve(n);

    }
```

```
      //solve(n) is expected to print Acciojob n times
      public static void solve(int n)
    {
        /*
            base case
            If n is 0 this means you need to print 0 times
            so simply return
            */
            if(n==0)
            {
                  return;
            }
            // When you come to line 27 that means its not base case
            // We have to print Acciojob n times
            System.out.println("ACCIOJOB");
            // printed once now I need to print n-1 times more
            // So we call solve(n-1)
            solve(n-1);
            // Once I come here I know that solve(n-1) has worked
            // and printed Acciojob n-1 times and we had printed
            // Acciojob once when we were on line 29
            // Thus Acciojob is printed for n times
            // solve(n) terminates now
    }
}
```

Lecture 1+2 (Recursion on Integer)

Recursively Print Numbers

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();

            printtillN(N, 1);
```

```
    }

      public static void printtillN(int N, int i) {
            // i.....N
            if(i==N){
                  System.out.print(i + " ");
                  return;
            }
            System.out.print(i + " ");
            printtillN(N,i+1);
    }
}
```

Recursively Print Numbers in Reverse

```
import java.util.*;
public class Main {

      public static void printtillN(int N) {
         // write code here
            if(N==0)
            {
                  return;
            }
            // n.....1
            System.out.print(N + " ");
            printtillN(N-1);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N;
        N = sc.nextInt();
        printtillN(N);
        sc.close();
    }
}
```

Decrease Print Increase Print

```java
import java.io.*;
import java.util.*;

public class Main {
  public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
      int n = scn.nextInt();
    printDI(n);
  }

  public static void printDI(int n) {
    // your code here
        if(n==0)
        {
              return;
        }
        // n.....1 1.....n
        System.out.println(n);
        printDI(n-1);
        System.out.println(n);

  }
}
```

Factorial Recursively

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(factorial_recursive(n));
        sc.close();
    }

    public static long factorial_recursive(int n) {
```

```
        // your code here
            if(n==0)
            {
                  return 1;
            }
            long x = factorial_recursive(n-1);
            return n*x;
    }
}
```

Recursive Fibonacci

```
import java.util.*;
import static java.lang.Math.ceil;

public class Main {

      public static int fib(int n ){
        // write code here
            if(n==1){
                  return 0;
            }
            if(n==2){
                  return 1;
            }
            /*int x = fib(n-1);
            int y = fib(n-2);
            return x+y;*/
            return fib(n-1)+fib(n-2);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        System.out.println(fib(n));
    }
}
```

Power Calculation

```java
import java.util.*;

class Main {
	public static long xPowerN(int x, int n){
		//write code here
		if(n==0)
		{
			return 1;
		}
		return x * xPowerN(x,n-1);
	}
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		int x = sc.nextInt();
		int n = sc.nextInt();
		System.out.println(xPowerN(x, n));
	}
}
```

Optimised Power Calculation

```java
import java.io.*;
import java.util.*;

class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

       int X,N;
        X = sc.nextInt();
        N = sc.nextInt();

        System.out.println(power(X,N));
    }

      public static long power(int x, int y)
```

```
    {
            //Write code here
            if(y==0){
                  return 1;
            }
            long a = power(x,y/2);
            if(y%2==0){
                  return a*a;
            }
            else{
                  return a*a*x;
            }
    }
}
```

Print The Pattern

```
import java.io.*;
import java.util.*;

class Main
{

      public static void solve(int n)
    {
        // Your code here
            if(n==0){
                  return;
            }
            solve(n-1);
            // pattern of size n-1 has been printed and cursor is at line
number n
            /*for(int i=0;i<n;i++){
                  System.out.print("* ");
            }*/
            f(n);
            System.out.println();

    }
```

```
      public static void f(int n)
      {
            if(n==0){
                  return;
            }
            System.out.print("* ");
            f(n-1);
      }



    public static void main(String args[])throws IOException
    {
        Scanner sc = new Scanner(System.in);

        int n=sc.nextInt();
            solve(n);

    }
}
```

Tower Of Hanoi

```
import java.io.*;
import java.util.*;
class Main {
      static void toh(int N, int from, int to, int aux) {
        //Write code here
            if(N==0){
                  return;
            }
            toh(N-1,from,aux,to);
            System.out.println("move disk " + N + " from rod " + from + "
to rod " + to);
            toh(N-1,aux,to,from);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N;
```

```
        N = sc.nextInt();

        toh(N, 1, 3, 2);

        sc.close();
    }
}
```

Lecture 3 (Recursion on Array)

Find Number of Digits

https://course.acciojob.com/idle?question=14c3baa4-93a7-4706-9180-505cf8635f43

```
import java.util.*;

public class Main {
      static int numOfDigi(int n) {
        //Write code here
            if(n==0){
                  return 0;
            }
            return 1 + numOfDigi(n/10);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n;
        n = sc.nextInt();
        int result = numOfDigi(n);
        System.out.println(result);
        sc.close();
    }
}
```

Count1

https://course.acciojob.com/idle?question=f9f01de4-0294-49d9-b5fb-8bc59cf088ab

```
import java.util.*;

public class Main {
      static int count1(int n) {
        //Write code here
            if(n==0){
                  return 0;
            }
            if(n%10==1){
                  return 1 + count1(n/10);
            }
            else{
                  return count1(n/10);
            }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(count1(n));
        sc.close();
    }
}
```

Print Array Recursively

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];

        for(int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        PrintArray(arr, n);
    }
```

```
      public static void PrintArray(int[] arr, int n) {
      // Write your code here
            //f(arr,0);
            if(n==0){
                  return;
            }
            PrintArray(arr,n-1);
            System.out.print(arr[n-1] + " ");
    }
      /*public static void f(int []arr,int i)
      {
            // i........arr.length-1
            if(i==arr.length){
                  return;
            }
            System.out.print(arr[i] + " ");
            f(arr,i+1);
      }*/
}
```

Print Reverse Array Recursively

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];

        for(int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        PrintReverseArray(arr, n);
    }

      public static void PrintReverseArray(int[] arr, int n) {
        //Write your code
            // Print contents from position n (index n-1) to position 1
(idx 0)
            if(n==0){
```

```
                return;
            }
            System.out.print(arr[n-1] + " ");
            PrintReverseArray(arr,n-1);
    }

  }
```

Smallest Number in an Array using Recursion

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        sc.close();
        System.out.println(recforMin(arr, n));
    }

      public static int recforMin(int[] arr, int n) {
        //Write your code here
            // It will tell min in array from pos 1 to pos n
            if(n==1)
            {
                return arr[0];
            }
            // 1.........n-1
            int a = recforMin(arr,n-1);
            return Math.min(a,arr[n-1]);
    }
}
```

Check if Array is Palindrome

https://course.acciojob.com/idle?question=4fd07c50-1cdf-488e-befd-e68b4b371c24

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        sc.close();

        System.out.println(isPalindrome(arr, n));
    }

      public static boolean isPalindrome(int []arr, int n)
      {
            int ans = isPalindromic(arr,0,n-1);
            if(ans==1){
                  return true;
            }
            return false;
      }

    public static int isPalindromic(int[] arr, int begin, int end) {
    // Write your code here
            // begin.....end  is a palindrome or not ?
            // 1 -> If yes
            // 0 -> If no
            if(begin>end){
                  return 1;
            }
            // begin<=end
            if(arr[begin]!=arr[end])
            {
                  return 0;
            }
            return isPalindromic(arr,begin+1,end-1);
    }
}
```

First Occurence Index

```java
import java.util.Scanner;
class Main {
	static Scanner s = new Scanner(System.in);
	public static int[] takeInput(){
		int N = s.nextInt();
		int[] A = new int[N];
		for(int i = 0; i < N; i++){
			A[i] = s.nextInt();
		}
		return A;
	}

	public static void main(String[] args) {
		int[] A = takeInput();
		int T = s.nextInt();
		System.out.println(firstIndex(A, T, 0));
	}

	static int firstIndex(int A[],int T,int startIndex)
	{
		//Write your code here
		if(startIndex==A.length)
		{
			return -1;
		}
		// startIndex is valid
		if(A[startIndex]==T){
			return startIndex;
		}
		return firstIndex(A,T,startIndex+1);
	}
}
```

Last Occurence Index

```java
import java.util.Scanner;
class Main {
```

```
      public static void main(String[] args) {
            Scanner s = new Scanner(System.in);
        int N = s.nextInt();
            int[] A = new int[N];
            for(int i = 0; i < N; i++){
                  A[i] = s.nextInt();
            }
            int T = s.nextInt();
            System.out.println(lastIndex(A, T, N-1));
      }

      static int lastIndex(int A[],int T,int startIndex)
      {
            //Write your code here
            if(startIndex==-1){
                  return -1;
            }
            if(A[startIndex]==T){
                  return startIndex;
            }
            return lastIndex(A,T,startIndex-1);
      }
}
```

Find Indices

```
import java.io.*;
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n,x;
        n = sc.nextInt();
        x = sc.nextInt();
        int arr1[] = new int[n];
        for(int i=0;i<n;i++)
            arr1[i] = sc.nextInt();
        findX(arr1, n, x);
    }
    static void findX(int arr[], int n, int x)
```

```
    {
        // write code here
            // will print all occurences of x starting from position 1 to
position n
            // base case
            if(n==0){
                  return;
            }
            // n is valid position
            // print occurences frompos 1 to n-1
            findX(arr,n-1,x);
            if(arr[n-1]==x){
                  System.out.print((n-1) + " ");
            }
    }
}
```

Lecture 4 (Recursion on String And 2D Arrays)

Print all Subsequences Of a String

```
import java.util.*;
import java.util.Scanner;

public class Main{
    static void printSubsequence(String s) {
        //Write your code here
            f(s,"");
    }
      static void f(String ques,String asf)
      {
            if(ques.length()==0){
                  System.out.print(asf + " ");
                  return;
            }
            char c = ques.charAt(0);
            String roq = ques.substring(1);
```

```
            f(roq,asf+c);
            f(roq,asf);
      }
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        printSubsequence(s);
    }
}
```

Aliter:

```
import java.util.*;
import java.util.Scanner;

public class Main{
    static void printSubsequence(String s) {
        //Write your code here
            f(s,"",0);
    }
      static void f(String ques,String asf, int idx)
      {
            if(idx==ques.length()){
                  System.out.print(asf + " ");
                  return;
            }
            char c = ques.charAt(idx);
            f(ques,asf+c,idx+1);
            f(ques,asf,idx+1);
      }
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        printSubsequence(s);
    }
}
```

No X

```java
import java.util.*;

public class Main {
    static String noX(String s) {
        // Write your code here
            if(s.length()==0){
                  return "";
            }
            if(s.charAt(0)=='x'){
                  return noX(s.substring(1));
            }
            else{
                  return s.charAt(0) + noX(s.substring(1));
            }

    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String s = sc.nextLine();
        System.out.println(noX(s));
    }
}
```

You can also use the index passing way as we did in the previous question

Print Stair Path

```java
import java.io.*;
import java.util.*;

public class Main {
    static void printStairPaths(int n,String pathSoFar) {
        //Write your code here
            if(n<0)
            {
                  return;
            }
```

```
            if(n==0){
                  System.out.println(pathSoFar);
                  return;
            }
            // n steps to climb
            // Lets take a 1 step jump
            printStairPaths(n-1,pathSoFar+"1");
            printStairPaths(n-2,pathSoFar+"2");
            printStairPaths(n-3,pathSoFar+"3");
    }
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        int n;
        n = sc.nextInt();
        printStairPaths(n,"");
    }
}
```

Aliter:

```
import java.io.*;
import java.util.*;

public class Main {
    static void printStairPaths(int n,String pathSoFar) {
        //Write your code here

            if(n==0){
                  System.out.println(pathSoFar);
                  return;
            }
            // n steps to climb
```

```
            // Lets take a 1 step jump
            if(n>=1)
            {
                  printStairPaths(n-1,pathSoFar+"1");
            }
            if(n>=2)
            {
                  printStairPaths(n-2,pathSoFar+"2");
            }
            if(n>=3)
            {
                  printStairPaths(n-3,pathSoFar+"3");
            }
    }
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        int n;
        n = sc.nextInt();
        printStairPaths(n,"");
    }
}
```

Keypad Combination

```
import java.io.*;
import java.util.*;

public class Main {
    static void printKPC(String ques) {
        //Write your code here
            String []ref =
{".;","abc","def","ghi","jkl","mno","pqrs","tu","vwx","yz"};
            f(ques,"",ref);
    }

      static void f(String ques,String asf,String[] ref)
      {
            if(ques.length()==0)
            {
                  System.out.println(asf);
```

```
                return;
            }
            // "784"
            char c = ques.charAt(0);// '7'
            String inter = ref[c-'0'];
            for(int i=0;i<inter.length();i++)
            {
                // inter.charAt(i)
                f(ques.substring(1),asf+inter.charAt(i),ref);

            }
      }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        String str;
        str = sc.nextLine();
        printKPC(str);
    }
}
```

Lecture 5 (Recursion on String And 2D Arrays)

Permutation Printing

```
import java.util.*;

public class Main{

   public static void main(String[] args) {
      Scanner scn = new Scanner(System.in);
      String s=scn.nextLine();
      String asf="";
      permutationPrint(s,asf);
   }
   public static void permutationPrint(String ques, String asf)
    {
        // your code here
```

```
            if(ques.length()==0){
                  System.out.println(asf);
                  return;
            }
            // ques = "bcxy"
            for(int i=0;i<ques.length();i++)
            {
                  char c = ques.charAt(i);
                  String left= ques.substring(0,i);
                  String right = ques.substring(i+1);
                  permutationPrint(left+right,asf+c);
            }
    }
  }
```

String Encodings

```
import java.util.*;

public class Main {
    public static void printEncodings(String str) {
        // Write your code here
            ArrayList<String> arr = new ArrayList<>();
            f(str,"",arr);
            Collections.sort(arr);
            for(int i=0;i<arr.size();i++)
            {
                  System.out.println(arr.get(i));
            }

    }
      public static void f(String ques, String asf, ArrayList<String>arr)
      {
            if(ques.length()==0)
            {
                  arr.add(asf);
                  return;
            }
            if(ques.charAt(0)=='0')
            {
```

```
                return;
            }
            char c = ques.charAt(0);
            int integer_equiv = c-'0';
            char mapping = (char)('a' + integer_equiv -1);
            f(ques.substring(1),asf+mapping,arr);
            if(ques.length()>=2)
            {
                int equiv = Integer.parseInt(ques.substring(0,2));
                if(equiv<=26)
                {
                    char mapp = (char)('a' + equiv -1);
                    f(ques.substring(2),asf+mapp,arr);
                }
            }
      }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();
        printEncodings(str);
    }
}
```

aMaze Paths

```
import java.util.*;
import java.io.*;
public class Main {
    public static void aMazePaths(int n, int m, String psf,int i , int j){
       //Write your code here
            if(i==n && j==m){
                System.out.println(psf);
                return;
            }
            if(i>n || j>m){
                return;
            }
            aMazePaths(n,m,psf+"h",i,j+1);
            aMazePaths(n,m,psf+"v",i+1,j);
```

```
    }

    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String[] inputLine;
        inputLine = br.readLine().trim().split(" ");
        int n = Integer.parseInt(inputLine[0]);
        inputLine = br.readLine().trim().split(" ");
        int m = Integer.parseInt(inputLine[0]);
        aMazePaths(n, m, "",1,1);
  }
}
```

Maze Problem

```
import java.io.*;
import java.util.*;

public class Main {
      public static void printMazePaths(int sr, int sc, int dr, int dc,
String psf) {
        //Write your code here
            if(sr==dr && sc==dc)
            {
                  System.out.println(psf);
                  return;
            }
            /*if(sr>dr || sc>dc){
                  return;
            }*/
            // horizontal jumps
            // (dc-sc) max horizontal jump length
            for(int i=1;i<=dc-sc;i++)
            {
                  printMazePaths(sr,sc+i,dr,dc,psf + "h" + i);
            }
            // vertical jumps
            // (dr-sr)
            for(int i=1;i<=dr-sr;i++)
```

```
            {
                  printMazePaths(sr+i,sc,dr,dc,psf+"v"+i);
            }
            // diagonal jump
            //min(dc-sc,dr-sr)
            for(int i=1;i<=Math.min(dr-sr,dc-sc);i++){
                  printMazePaths(sr+i,sc+i,dr,dc,psf+"d" +i);
            }
    }
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int n = Integer.parseInt(br.readLine());
        int m = Integer.parseInt(br.readLine());
        printMazePaths(0, 0, n - 1, m - 1, "");
    }
}
```

Contest Discussion (10 Feb)

Fun with Strings

https://course.acciojob.com/idle?question=4a0ab963-b31e-4e17-9870-f30d64ef6eae

```
import java.io.*;
import java.util.*;
class Main
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);

        String s=sc.nextLine();

        Solution obj=new Solution();
        System.out.println(obj.solve(s));
        sc.close();
    }
}
```

```java
class Solution{
    public String solve(String s)
    {
        // your code here
            /*if(s.length()==1)
            {
                  return s;
            }
            return solve(s.substring(1)) + s.charAt(0);
        */
            char []arr = s.toCharArray();
            f(0,arr.length-1,arr);
            String t = new String(arr);
            return t;
    }
      public void f(int start, int end, char []arr)
      {
            //[start.....end]
            if(start>end){
                  return;
            }
            // start<=end
            char temp=arr[start];
            arr[start]=arr[end];
            arr[end]=temp;
            f(start+1,end-1,arr);
      }
}
```

Power Of 5

https://course.acciojob.com/idle?question=e4df1316-d52e-4512-a26c-311bb9eda650

```java
import java.util.*;

class Solution {
    public String powOf5(int n) {
        // write code here
            if(n==1){
                  return "Yes";
```

```
            }
            if(n%5==0)
            {
                  return powOf5(n/5);
            }
            return "No";
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        Solution Obj =  new Solution();
        System.out.print(Obj.powOf5(n));
    }
}
```

Catching Up

https://course.acciojob.com/idle?question=39a7cdff-37a9-4233-821a-42b418bf3463

```
import java.util.*;

class Solution {

    int catchingUp(int a, int b) {
       // write code here
            if(a>b)
            {
                  int temp=a;
                  a=b;
                  b=temp;
            }
            if(a==0){
                  return b;
            }
            // a<=b guaranteed
            if(b%a==0)
            {
```

```
                return a;
            }
            return catchingUp(b%a,a);
    }

}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a= sc.nextInt();
        int b= sc.nextInt();
        Solution Obj = new Solution();
        System.out.println(Obj.catchingUp(a,b));

    }
}
```

Linked List

Node

```
public class Node {
      int data;
      Node next;
}
```

Linked List

```
public class LinkedList {
	Node head;

	void insert(int n)
	{
		Node new_node = new Node();
		new_node.data=n;

		if(head==null) {
			head=new_node;
			return;
		}

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

	}

	void insertAtFront(int n) {
		Node new_node = new Node();
		new_node.data=n;
		new_node.next=head;
		head=new_node;
	}

	void insertAfterValue(int value,int insertvalue) {
		Node temp= head;
		while(temp.data!=value) {
			temp=temp.next;
		}
		Node new_node = new Node();
		new_node.data=insertvalue;
		new_node.next=temp.next;
		temp.next=new_node;
		return;
	}

	void deleteFront() {
```

```
		if(head==null) {
			return;
		}
		head=head.next;
	}

	void deleteValue(int n)
	{
		if(head==null) {
			return;
		}
		else if(head.data==n) {
			head=head.next;

		}
		else {
			Node curr = head;
			Node prev = null;
			while(curr.data!=n) {
				prev=curr;
				curr=curr.next;
			}
			prev.next=curr.next;
			curr=null;

		}
		return;

	}

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

}
```

Runner

```java
public class Runner {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		LinkedList ll = new LinkedList();
		ll.insert(5);
		ll.insert(18);
		//ll.show();
		ll.insert(15);
		ll.insertAtFront(4);
		ll.show();
		System.out.println();
		ll.insertAfterValue(18, -5);
		ll.show();
		ll.deleteValue(18);
		System.out.println();
		ll.show();
		System.out.println();
		ll.deleteFront();
		ll.show();
		ll.deleteValue(15);
		System.out.println();
		ll.show();
	}
}
```

Delete a Node

https://course.acciojob.com/idle?question=7b346e14-d70f-4dc0-a82f-f3c2f3da9646

```java
class Solution {
    public static void remove(LinkedList ll, int toRemove){
		if(toRemove == 0){
			ll.head=ll.head.next;
		}
		else{
			Node curr=ll.head;
			Node prev = null;
			for(int i=0;i<toRemove;i++){
				prev=curr;
				curr=curr.next;
			}
```

```
                prev.next=curr.next;
                curr=null;
            }
            return;
    }
}
```

Also think of adding (Idea discussed in class)

Middle Of linked list

Discussed 2 approaches

1. Two pass solution by counting nodes in linked list
2. Hare and Tortoise one pass algorithm (slow and fast pointer)

```
class Solution
{
    static Node midpointOfLinkedList(Node head)
    {
        //Your code here
            /*int n =0;
            Node temp =head;
            while(temp!=null){
                n++;
                temp=temp.next;
            }
            // n is the total nodes in ll
            temp=head;
            for(int i=0;i<n/2;i++){
                temp=temp.next;
            }
            return temp;
            */
            Node fast=head;
            Node slow=head;
            while(fast!=null && fast.next!=null){
                fast=fast.next.next;
                slow=slow.next;
            }
            return slow;
```

```
    }

}
```

Reverse a linked list

Reverse a linked list and return the reference of head of reversed linked list

https://leetcode.com/problems/reverse-linked-list/

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev =null;
        ListNode curr=head;
        while(curr!=null){
            ListNode temp=curr.next;
            curr.next=prev;
            prev=curr;
            curr=temp;
        }
        return prev;
    }
}
```

Print in reverse

Approach 1:
Reverse the actual linked list. Then after reversing print the contents of the linked list

```
public static Node revLL(Node head){
            Node prev = null;
            Node curr=head;
            while(curr!=null){
                  Node temp=curr.next;
                  curr.next=prev;
```

```
                prev=curr;
                curr=temp;
          }
          //prev is the head of reversed linked list
          return prev;
    }
   public static void reverse(Node curr){
          //revLL will reverse the linked list from curr and give
          // refrence of head of reversed ll that we are storing
          // in temp
          Node temp = revLL(curr);
          // once we have access of head of reverse ll now just print
          while(temp!=null){
                System.out.print(temp.data + " ");
                temp=temp.next;
          }


   }
```

Approach 2:
Use recursion to print the linked list from end. This does not alter the structure of the linked list.

```
public static void reverse(Node curr){
      // write code here
          if(curr==null){
                return;
          }
          reverse(curr.next);
          System.out.print(curr.data + " ");
   }
```

Remove nth node from the end

Approach 1:
Count the number of nodes in ll. Nth node fromend is size-n+1 th node from start. So we need to move pointer size-n times to reach the node that needs to be deleted.

If you are not using dummy node then you need to handle head deletion case separately

```
public int size(Node head)
      {
            int count =0;
            Node temp=head;
            while(temp!=null){
                  count++;
                  temp=temp.next;
            }
            return count;
      }
    public static Node removeNthFromEnd(Node head, int n) {

            Node dummy=new Node(-1);
            dummy.next=head;
            int size = size(head);
            Node prev = dummy;
            Node curr=head;
            for(int i=0;i<size-n;i++){
                  prev=curr;
                  curr=curr.next;
            }
            prev.next=curr.next;
            return dummy.next;
}
```

Approach 2:
Two pointer approach

```
public static Node removeNthFromEnd(Node head, int n) {
```

```
            // creating a dummy node to avoid handling special case of head
            Node dummy = new Node(-1);
            dummy.next=head;
            Node front=dummy;
            Node back=dummy;
            // giving a headstart to the front pointer to create a gap of n
            for(int i=0;i<n;i++){
                  front=front.next;
            }
            // moving together till front reaches the last node
            while(front.next!=null){
                  front=front.next;
                  back=back.next;
            }
            // back is previous to the node that needs to be deleted
            back.next=back.next.next;
            // dummy.next is the head
            return dummy.next;
}
```

Merge 2 sorted linked list

Approach1: Create a new linked list (Extra Space)

Had done this on LeetCode

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(-1);
        ListNode prev = dummy;
        ListNode l1 = list1;
        ListNode l2 =list2;
        while(l1!=null && l2!=null){
            if(l1.val<l2.val)
            {
                prev.next= new ListNode(l1.val);
                prev=prev.next;
                l1=l1.next;
```

```
            }
            else{
                ListNode t = new ListNode(l2.val);
                prev.next=t;
                prev=prev.next;
                l2=l2.next;
            }
        }

        while(l1!=null){
            prev.next=new ListNode(l1.val);
            prev=prev.next;
            l1=l1.next;
        }
        while(l2!=null){
            prev.next=new ListNode(l2.val);
            prev=prev.next;
            l2=l2.next;
        }
        return dummy.next;
    }
}
```

Approach 2: Splice or manipulate the links

```
class Solution {
 static Node merge(Node x, Node y){
       // Write your code here
       Node dummy = new Node(-1);
       Node prev = dummy;
       Node l1 = x;
       Node l2 = y;
       while(l1!=null && l2!=null){
             if(l1.data<l2.data){
                   prev.next=l1;
                   prev=prev.next;
                   l1=l1.next;
             }
             else{
```

```
                    prev.next=l2;
                    prev=prev.next;
                    l2=l2.next;
              }
        }
        if(l1!=null){
              prev.next=l1;
        }
        else{
              prev.next=l2;
        }
        return dummy.next;
    }
}
```

Linked List Cycle

Slow and fast pointer approach (Floyd's cycle detection algorithm)

https://leetcode.com/problems/linked-list-cycle/
https://course.acciojob.com/idle?question=130e273c-3811-43a0-95e0-2862dbd39118

```
public static boolean detectLoop(Node head){
   //your code here
           Node fast = head;
           Node slow=head;
           while(fast!=null && fast.next!=null){
                 fast=fast.next.next;
                 slow=slow.next;
                 if(slow==fast){
                       return true;
                 }
           }
           return false;
    }
```

Linked List Cycle 2

```
public static Node detectLoop(Node head){
        //write code here
            Node fast=head;
            Node slow=head;
            while(fast!=null && fast.next!=null)
            {
                  fast=fast.next.next;
                  slow=slow.next;
                  if(fast==slow)
                  {
                        break;
                  }
            }
            if(fast==slow)
            {
                  // cycle
                  fast=head;
                  while(fast!=slow)
                  {
                        fast=fast.next;
                        slow=slow.next;
                  }
                  return fast;
            }
            return null;
    }
```

Intersection of 2 Linked List

https://course.acciojob.com/idle?question=f7994df4-2130-46b1-a9c2-3b3c68b7366c

https://leetcode.com/problems/intersection-of-two-linked-lists/

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        ListNode l1 = headA;
          ListNode l2 = headB;
          while(l1!=l2)
          {
```

```
                l1 = (l1==null)?headB:l1.next;
                if(l2!=null){
                      l2=l2.next;
                }
                else{
                      l2=headA;
                }
          }
          return l2;
    }
```

Add 2 numbers represented by linked lists

```
public static Node reverse(Node head)
      {
            Node prev = null;
            Node curr=head;
            while(curr!=null){
                  Node temp=curr.next;
                  curr.next=prev;
                  prev=curr;
                  curr=temp;
            }
            return prev;

      }

    public static Node addTwoLinkedLists(Node head1, Node head2)
    {
        //Write your code here
            Node l1 = reverse(head1);
            Node l2 = reverse(head2);
            Node dummy = new Node(-1);
            Node t = dummy;
            int carry=0;
            int sum;
            while(l1!=null || l2!=null)
            {
                  sum=carry;
                  if(l1!=null){
```

```
                        sum+=l1.data;
                        l1=l1.next;
                }
                if(l2!=null){
                        sum+=l2.data;
                        l2=l2.next;
                }
                t.next=new Node(sum%10);
                t=t.next;
                carry=sum/10;
        }

        if(carry==1){
                t.next=new Node(1);
        }

        Node ans = reverse(dummy.next);
        return ans;

    }
```

Subtract Linked Lists

```
static int size(Node head)
      {
            int count =0;
            Node temp = head;
            while(temp!=null){
                  count++;
                  temp=temp.next;
            }
            return count;
      }
      static Node reverse(Node head)
      {
            Node prev = null;
            Node curr = head;
            while(curr!=null){
                  Node temp=curr.next;
                  curr.next=prev;
```

```
                prev=curr;
                curr=temp;
          }
          return prev;

    }
    static int comp(Node ref1, Node ref2)
    {
          // this function will return 0
          // if ref1 amd ref2 linked list are same number
          // 1 if ref1 > ref2
          // -1 ref1 <ref2
          int size1 = size(ref1);
          int size2 = size(ref2);
          if(size1>size2){
                return 1;
          }
          else if(size1<size2){
                return -1;
          }
          else
          {
                Node temp1 = ref1;
                Node temp2 = ref2;
                while(temp1!=null && temp2!=null){
                      if(temp1.data>temp2.data){
                            return 1;
                      }
                      else if(temp1.data<temp2.data){
                            return -1;
                      }
                      else{
                            temp1=temp1.next;
                            temp2=temp2.next;
                      }
                }
                return 0;

          }
    }
```

```
    static Node subtract(Node l1, Node l2)
    {
            int status = comp(l1,l2);
            if(status==0)
            {
                  return new Node(0);
            }
            else
            {
                  if(status==-1)
                  {
                        Node temp = l2;
                        l2=l1;
                        l1=temp;
                  }

            }
            // l1 is pointing to bigger number
            l1= reverse(l1);
            l2 = reverse(l2);

            Node dummy = new Node(-1);
            Node t = dummy;
            int borrow=0;
            int sum;
            while(l1!=null || l2!=null)
            {
                  sum=0;
                  sum=sum-borrow;
                  if(l1!=null)
                  {
                        sum+=l1.data;
                        l1=l1.next;
                  }
                  if(l2!=null){
                        sum-=l2.data;
                        l2=l2.next;
                  }
                  if(sum<0){
                        sum+=10;
                        borrow=1;
                  }
                  else{
```

```
                        borrow=0;
                }
                t.next=new Node(sum);
                t=t.next;
            }

            Node rev = reverse(dummy.next);
            while(rev.data==0){
                rev=rev.next;
            }
            return rev;


    }
```

Segregating linked list of 0,1,2

```
static void unfold(Node head)
    {
            Node d0 = new Node(-1);
              Node d1 = new Node(-1);
              Node d2 = new Node(-1);
              Node t0 = d0;
              Node t1 = d1;
              Node t2 = d2;
              Node temp = head;
              while(temp != null){
                    if(temp.data == 0){
                          t0.next = temp;
                          t0 = t0.next;
                    }else if(temp.data == 1){
                          t1.next = temp;
                          t1 = t1.next;
                    }else{
                          t2.next = temp;
                          t2 = t2.next;
                    }
                    temp = temp.next;
               }
              // connection
```

```
            t0.next = d1.next;
            t1.next = d2.next;
            t2.next = null;
            head = d0.next;
            printLl(head);
    }

      public static void printLl(Node head){
            while(head != null){
                  System.out.print(head.data + " ");
                  head = head.next;
            }
            System.out.println();
      }
```

Remove Duplicates from sorted Linked List

```
public Node solve(Node head) {
        // your code here
            Node curr = head;
            while(curr!=null)
            {
                  while(curr.next!=null && curr.data==curr.next.data){
                        curr.next=curr.next.next;
                  }
                  curr=curr.next;
            }
            return head;
    }
```

Unfold Linked List

```
static void print(Node head)
      {
            Node temp=head;
            while(temp!=null){
```

```
                System.out.print(temp.data + " ");
                temp=temp.next;
          }
    }

    static Node reverse(Node head)
    {
          Node prev = null;
          Node curr = head;
          while(curr!=null){
                Node temp=curr.next;
                curr.next=prev;
                prev=curr;
                curr=temp;
          }
          return prev;

    }
  static void unfold(Node head)
  {
   // Your code here
          Node temp=head;
          Node d1 = new Node(-1);
          Node d2 = new Node(-1);
          Node l1=d1;
          Node l2 =d2;
          int count=0;
          while(temp!=null)
          {
                if(count%2==0)
                {
                      l1.next=temp;
                      l1=l1.next;
                      temp=temp.next;
                }
                else
                {
                      l2.next=temp;
                      l2=l2.next;
                      temp=temp.next;
                }
                count++;
          }
```

```
            l1.next=null;
            l2.next=null;

            Node rev = reverse(d2.next);
            l1.next=rev;
            print(d1.next);
    }
```

Merge Sort And Quick Sort

Implementing MergeSort

```
class Solution {
static void mergeSort(int[] arr,int l,int r)
      {
     // Your code here
             if(l<r)
             {
                  int mid = (l+r)/2;
                mergeSort(arr,l,mid);
              mergeSort(arr,mid+1,r);
              merge(arr,l,mid,r);
            }
    }
      static void merge(int []arr,int l, int mid, int r){
          // l......mid
          // mid+1....r
          int []b = new int[r-l+1];
          int i=l;
          int j=mid+1;
          int k=0;
          while(i<=mid && j<=r)
          {
              if(arr[i]<=arr[j]){
```

```
                b[k]=arr[i];
                i++;
                k++;
            }
            else{
                b[k]=arr[j];
                j++;
                k++;
            }
        }
        while(i<=mid){
            b[k]=arr[i];
            i++;
            k++;
        }
        while(j<=r){
            b[k]=arr[j];
            j++;
            k++;
        }
        // b
        k=0;
        for(int p=l;p<=r;p++)
        {
            arr[p]=b[k];
            k++;

        }
    }
}
```

Implementing QuickSort

You can choose any element as pivot. Some random element / element at l/ element at r
In the below implementation we have taken arr[r] to be the pivot element

```
class Solution{
    static void quickSort(int[] arr){
      //Write code here
          int n = arr.length;
          qs(arr,0,n-1);
    }
```

```
	static void qs(int []arr, int l, int r)
	{
		if(l<r)
		{
			int pI = partition(arr,l,r);
			qs(arr,l,pI-1);
			qs(arr,pI+1,r);
		}
	}

	static int partition(int []arr, int left ,int right)
	{
		//l....r
		int pivot = arr[right];
		int i=left;
		//int j=l;
		for(int j=left;j<right;j++){
			if(arr[j]<=pivot)
			{
				int t= arr[j];
				arr[j]=arr[i];
				arr[i]=t;
				i++;
			}
		}
		arr[right]=arr[i];
		arr[i]=pivot;

		return i;

	}
}
```

Linked List

MergeSort on Linked List

```
public static Node merge(Node head1, Node head2)
      {
            Node dummy = new Node(-1);
            Node t = dummy;
            Node t1=head1;
            Node t2 = head2;
            while(t1!=null && t2!=null){
                  if(t1.data<=t2.data){
                        t.next=t1;
                        t=t.next;
                        t1=t1.next;
                  }
                  else{
                        t.next=t2;
                        t2=t2.next;
                        t=t.next;
                  }
            }
            if(t1!=null){
                  t.next=t1;
            }
            else{
                  t.next=t2;
            }
            return dummy.next;

      }
      public static Node getMid(Node head){
            Node slow=head;
            Node fast=head;
            while(fast.next!=null && fast.next.next!=null){
                  fast=fast.next.next;
                  slow=slow.next;
            }
            return slow;
      }
    public static Node mergesort(Node head){
        //Write your code here
            if(head==null || head.next==null){
                  return head;
            }
            Node temp=head;
            Node mid = getMid(head);
```

```
            Node head2=mid.next;
            mid.next=null;
            Node lHead = mergesort(head);
            Node rHead = mergesort(head2);
            Node ans = merge(lHead,rHead);
            return ans;

    }
```

QuickSort on Linked List

```
public static Node[] partition(Node head)
      {
            Node pivot = head;
            Node d1 = new Node(-1);
            Node d2 = new Node(-1);
            Node l1=d1;
            Node l2 =d2;
            Node temp=head;
            while(temp!=null)
            {
                  if(temp!=pivot)
                  {
                        if(temp.data<=pivot.data)
                        {
                              l1.next=temp;
                              l1=l1.next;
                              //temp=temp.next;
                        }
                        else{
                              l2.next=temp;
                              l2=l2.next;
                              //temp=temp.next;
                        }
                  }
                  temp=temp.next;
            }
            l1.next=null;
            l2.next=null;
            pivot.next=null;
```

```
            Node []arr = {d1.next,pivot,d2.next};
            return arr;

      }
    public static Node quickSort(Node node)
    {
        //Your code here
            if(node == null || node.next==null){
                  return node;
            }

            Node[]arr = partition(node);
            // arr[0]-> left
            // arr[1] -> pivot
            //arr[2] -> right
            Node hleft = quickSort(arr[0]);
            Node hright = quickSort(arr[2]);
            arr[1].next=hright;
            if(hleft==null){
                  return arr[1];
            }
            Node temp =hleft;
            while(temp.next!=null){
                  temp=temp.next;
            }
            temp.next=arr[1];
            return hleft;


    }
```

Flattening a Linked List

```
static Node flatten(Node root) {
    // your code here
        if(root.right==null){
              return root;
        }
        Node t = flatten(root.right);
        Node mHead = merge(root,t);
```

```
        return mHead;
  }

      static Node merge(Node head1, Node head2)
      {
            Node d = new Node(-1);
            Node prev = d;
            Node l1 = head1;
            Node l2 =head2;
            l1.right=null;
            while(l1!=null && l2!=null){
                  if(l1.data<=l2.data){
                        prev.down=l1;
                        l1=l1.down;
                        prev=prev.down;
                  }
                  else{
                        prev.down=l2;
                        l2=l2.down;
                        prev=prev.down;
                  }
            }
            if(l1!=null){
                  prev.down=l1;
            }
            else{
                  prev.down=l2;
            }
            return d.down;
      }
```

Copy LinkedList with random pointer

```
Node copyList(Node head) {
        // your code here
            Node curr = head;
            while(curr!=null)
            {
                  Node temp = curr.next;
                  Node t = new Node(curr.data);
```

```
                t.next=temp;
                curr.next=t;
                curr=temp;
            }
            curr = head;
            while(curr!=null)
            {
                if(curr.random==null)
                {
                    curr.next.random=null;
                }
                else
                {
                    curr.next.random=curr.random.next;
                }
                curr=curr.next.next;
            }
            Node original = head;
            Node copy = head.next;
            Node temp = copy;

            while(original!=null){
                original.next=original.next.next;
                if(copy.next!=null)
                {
                    copy.next=copy.next.next;
                }
                original=original.next;
                copy=copy.next;
            }
            return temp;


    }
```

Rotate List

```
static int size(Node head){
            Node temp=head;
            int count =0;
```

```
            while(temp!=null){
                  count++;
                  temp=temp.next;
            }
            return count;
      }

      static void rotateRight(Node head, int k) {
    // Your code here
            int n = size(head);
            k = k%n;
            if(k!=0)
            {
                  Node curr =head;
                  Node prev=null;
                  for(int i=0;i<n-k;i++){
                        prev=curr;
                        curr=curr.next;
                  }
                  prev.next=null;
                  Node t = curr;
                  while(t.next!=null){
                        t=t.next;
                  }
                  t.next=head;
                  head=curr;
            }
            Node temp2 = head;
            while(temp2!=null){
                  System.out.print(temp2.data + " ");
                  temp2=temp2.next;
            }
    }
```

Insertion in Doubly Linked List

https://course.acciojob.com/idle?question=c44249e8-0776-4bcf-883d-fbdad0c956a9

```
static Node insertAtHead(Node head,int K) {
```

```
        // Write your code here.
            Node nn = new Node(K);
            if(head==null)
            {
                  head=nn;
                  return head;
            }
            nn.next=head;
            head.prev=nn;
            head=nn;
            return head;
    }
```

Delete node in doubly linked list

https://course.acciojob.com/idle?question=38362321-9064-4578-99ea-bf81a8f3ef0e

```
import java.util.*;

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

class DoublyLinkedList {
    Node head;
    Node tail;
    int length;

    DoublyLinkedList() {
        this.head = null;
        this.tail = null;
        this.length = 0;
```

```
    }

    void append(int val) {
        Node newNode = new Node(val);
        if (this.length == 0) {
            this.head = newNode;
            this.tail = newNode;
        } else {
            this.tail.next = newNode;
            newNode.prev = this.tail;
            this.tail = newNode;
        }
        this.length++;
    }

    void prepend(int val) {
        Node newNode = new Node(val);
        if (this.length == 0) {
            this.head = newNode;
            this.tail = newNode;
        } else {
            this.head.prev = newNode;
            newNode.next = this.head;
            this.head = newNode;
        }
        this.length++;
    }

    void delete(Node node) {
        // Write your code here
            if(head==null){
                  return;
            }
            if(node==head && node==tail){
                  head=null;
                  tail=null;
                  return;
            }
            if(node==head){
                  head=head.next;
                  head.prev=null;
                  return;
            }
```

```
            Node p = node.prev;
            p.next=node.next;
            if(node.next!=null)
            {
                  node.next.prev=p;
            }
            else{
                  tail=p;
            }
            return;
    }

    void printList() {
        Node curr = this.head;
        while (curr != null) {
            System.out.print(curr.val + " ");
            curr = curr.next;
        }
        System.out.println();
    }
}

class Solution{
    public static DoublyLinkedList deleteNodeWithX(DoublyLinkedList
linkedList, int x){
        //Write your code here
            Node head = linkedList.head;
            Node temp=head;
            while(temp!=null){
                  if(temp.val==x){
                        linkedList.delete(temp);
                  }
                  temp=temp.next;
            }
            return linkedList;


    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N= sc.nextInt();
```

```
        int M= sc.nextInt();
        DoublyLinkedList linkedList = new DoublyLinkedList();
        for(int i=0; i<N; i++){
            linkedList.append(sc.nextInt());
        }
        Solution Obj = new Solution();
        linkedList = Obj.deleteNodeWithX(linkedList,M);
        linkedList.printList();

    }
}
```

Flatten multi level doubly linked list

Try this using iteration also

```
public static Node flatten(Node head, Node rest) {
       //Write your code here
           flat(head);
           return head;
   }

     public static void flat(Node head)
     {
           Node temp=head;
           while(temp!=null)
           {
                 if(temp.child!=null)
                 {
                       flat(temp.child);
                       Node p = temp.child;
                       Node next = temp.next;
                       temp.next=p;
                       p.prev=temp;
                       Node tt = temp;
                       while(tt.next!=null){
                             tt=tt.next;
                       }
                       tt.next=next;
                       if(next!=null)
```

```
                        {
                              next.prev=tt;
                        }
                        temp.child=null;
                        temp=next;
                  }
                  else
                  {
                        temp=temp.next;
                  }
            }
            return;

      }
```

Design Browser History

```
class BrowserHistory {
      Node head;
      Node curr;
      Node tail;
    public BrowserHistory(String homepage) {
        // your code here
            Node n = new Node(homepage);
            head=n;
            curr=n;
            tail=n;
    }
    public void visit(String url) {
        // your code here
            Node new_node = new Node(url);
            curr.next=new_node;
            new_node.prev=curr;
            curr=new_node;
            tail=new_node;
    }
    public void back(int steps) {
        // your code here
            while(curr!=head && steps>0){
                  curr=curr.prev;
```

```
                steps--;
            }
            System.out.println(curr.website);

    }
    public void forward(int steps) {
        // your code here
            while(curr!=tail && steps>0){
                curr=curr.next;
                steps--;
            }
            System.out.println(curr.website);

    }
}
```

Insertion in Circular Linked List

```
import java.util.*;
import java.io.*;

class Node{
      int data;
      Node next;
      Node(int d){
            data=d;
      }
}

class CircularLL{
      Node tail;

      void insertAtEnd(int val)
      {
            Node n = new Node(val);
            if(tail==null)
            {
                  n.next=n;
                  tail=n;
                  return;
```

```
            }

            n.next=tail.next;
            tail.next=n;
            tail=n;
      }

      void print(){
            Node curr = tail.next;
            while(curr!=tail){
                  System.out.print(curr.data + " ");
                  curr=curr.next;
            }
            System.out.print(curr.data);
      }


}

public class Main {
    public static void main(String args[]) {
        //your code here
            Scanner sc = new Scanner(System.in);
            int n = sc.nextInt();
            CircularLL ll = new CircularLL();
            for(int i=0;i<n;i++){
                  ll.insertAtEnd(sc.nextInt());
            }
            ll.insertAtEnd(sc.nextInt());
            ll.print();
    }
}
```

Partitioning a linked list

```
public void partition(Node head, int x) {
        // Your code here
            Node d1=new Node(-1);
            Node d2= new Node(-1);
            Node l1 = d1;
```

```
            Node l2 =d2;
            Node temp=head;
            while(temp!=null){
                  if(temp.data<x){
                        l1.next=temp;
                        l1=l1.next;
                  }
                  else{
                        l2.next=temp;
                        l2=l2.next;
                  }
                  temp=temp.next;
            }
            l2.next=null;
            l1.next=null;
            l1.next=d2.next;
            Node t = d1.next;
            while(t!=null){
                  System.out.print(t.data + " ");
                  t=t.next;
            }
      }
```