**JAVA STDIN AND STDOUT**

***Most HackerRank challenges require you to read input from [stdin](https://en.wikipedia.org/wiki/Standard_streams" \l "Standard_input_.28stdin.29) (standard input) and write output to [stdout](https://en.wikipedia.org/wiki/Standard_streams" \l "Standard_output_.28stdout.29) (standard output).***

***One popular way to read input from stdin is by using the***[***Scanner class***](https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html)***and specifying the Input Stream as System.in.***

***For example:***

***Scanner scanner = new Scanner(System.in);***

***String myString = scanner.next();***

***int myInt = scanner.nextInt();***

***scanner.close();***

***System.out.println("myString is: " + myString);***

***System.out.println("myInt is: " + myInt);***

***The code above creates a Scanner object named scanner  and uses it to read a String and an int. It then closes the Scanner object because there is no more input to read, and prints to stdout using System.out.println(String). So,***

**if our input is:**

***Hi 5***

**Our code will print:**

***myString is: Hi***

***myInt is: 5***

***Alternatively, you can use the [BufferedReader class](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html).***

**Task: *In this challenge, you must read 3  integers from stdin and then print them to stdout. Each integer must be printed on a new line. To make the problem a little easier, a portion of the code is provided for you in the editor below.***

**Input Format:**

***There are 3  lines of input, and each line contains a single integer.***

**Sample Input:**

***42***

***100***

***125***

**Sample Output:**

***42***

***100***

***125***

**CODE:**

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

int a = scan.nextInt();

int b = scan.nextInt();

int c = scan.nextInt();

System.out.println(a);

System.out.println(b);

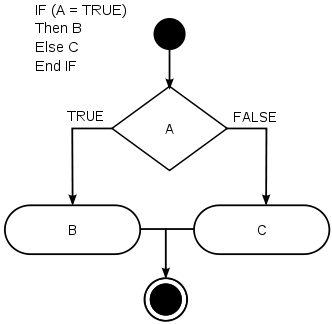
System.out.println(c);

}

}

**JAVA IF-ELSE**

***In this challenge, we test your knowledge of using if-else conditional statements to automate decision-making processes. An if-else statement has the following logical flow:***

******

***Source:***[***Wikipedia***](https://en.wikipedia.org/wiki/Conditional_%28computer_programming%29)

**Task: *Given an integer, ,n perform the following conditional actions:***

* ***If n is odd, print Weird***
* ***If  n is even and in the inclusive range of 2  to 5 , print Not Weird***
* ***If n  is even and in the inclusive range of  6 to 20 , print Weird***
* ***If n  is even and greater than  20, print Not Weird***

***Complete the stub code provided in your editor to print whether or not n  is weird.***

**Input Format:**

***A single line containing a positive integer n, .***

***Constraints***

* ***1<= n <= 100***

**Output Format:**

***Print Weird if the number is weird; otherwise, print Not Weird.***

***Sample Input 0***

***3***

***Sample Output 0***

***Weird***

***Sample Input 1***

***24***

***Sample Output 1***

***Not Weird***

**Explanation:**

***Sample Case 0:  n=3  
 n  is odd and odd numbers are weird, so we print Weird.***

***Sample Case 1:  n=24  
 n > 20 and n   is even, so it isn't weird. Thus, we print Not Weird.***

**CODE:**

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.regex.\*;

public class solution {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n = sc.nextInt();

String ans="";

if(n%2==1){

ans="Weird";

}

else{

if(n>= 6 && n <= 20)

{

ans="Weird";

}

else

{

ans="Not Weird";

}

}

System.out.println(ans);

}

}

**JAVA STDIN AND STDOUT II**

***In this challenge, you must read an integer, a double, and a String from stdin, then print the values according to the instructions in the Output Format section below. To make the problem a little easier, a portion of the code is provided for you in the editor.***

***Note: We recommend completing***[***Java Stdin and Stdout I***](https://www.hackerrank.com/challenges/java-stdin-and-stdout-1)***before attempting this challenge.***

**Input Format:**

***There are three lines of input:***

1. ***The first line contains an integer.***
2. ***The second line contains a double.***
3. ***The third line contains a String.***

**Output Format:**

***There are three lines of output:***

1. ***On the first line, print String: followed by the unaltered String read from stdin.***
2. ***On the second line, print Double: followed by the unaltered double read from stdin.***
3. ***On the third line, print Int: followed by the unaltered integer read from stdin.***

***To make the problem easier, a portion of the code is already provided in the editor.***

**Note*: If you use the nextLine() method immediately following the nextInt() method, recall that nextInt() reads integer tokens; because of this, the last newline character for that line of integer input is still queued in the input buffer and the next nextLine() will be reading the remainder of the integer line (which is empty).***

**Sample Input:**

***42***

***3.1415***

***Welcome to HackerRank's Java tutorials!***

**Sample Output:**

***String: Welcome to HackerRank's Java tutorials!***

***Double: 3.1415***

***Int: 42***

**CODE:**

import java.util.Scanner;

public class Solution {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

int i = scan.nextInt();

Double d = scan.nextDouble();

scan.nextLine();

String s = scan.nextLine();

System.out.println("String: " + s);

System.out.println("Double: " + d);

System.out.println("Int: " + i);

}

}

**JAVA OUTPUT FORMATING**

***Java's*System.out.printf*function can be used to print formatted output. The purpose of this exercise is to test your understanding of formatting output using*printf*.***

***To get you started, a portion of the solution is provided for you in the editor; you must format and print the input to complete the solution.***

**Input Format:**

***Every line of input will contain a*String*followed by an*integer*.  
Each*String*will have a maximum of  10 alphabetic characters, and each*integer*will be in the inclusive range from  0 to 999.***

**Output Format:**

***In each line of output there should be two columns:  
The first column contains the*String*and is left justified using exactly 15  characters.  
The second column contains the*integer*, expressed in exactly 3  digits; if the original input has less than three digits, you must pad your output's leading digits with zeroes.***

**Sample Input:**

***java 100***

***cpp 65***

***python 50***

**Sample Output:**

***================================***

***java 100***

***cpp 065***

***python 050***

***================================***

**Explanation:**

***Each*String*is left-justified with trailing whitespace through the first 15  characters. The leading digit of the*integer*is the  16th character, and each*integer*that was less than 3  digits now has leading zeroes.***

**CODE:**

import java.util.Scanner;

public class Solution {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("================================");

for(int i=0;i<3;i++)

{

String s1=sc.next();

int x=sc.nextInt();

System.out.printf("%-15s%03d\n",s1,x);

}

System.out.println("================================");

}

}

**JAVA LOOPS I**

**Objective:** *In this challenge, we're going to use loops to help us do some simple math.*

*Task: Given an integer,N , print its first 10 multiples. Each multiple  (where 1<=i<=10 ) should be printed on a new line in the form: N x i = result.*

**Input Format:**

*A single integer, .*

**Constraints:**

* *2<=N<=20*

**Output Format:**

*Print  10 lines of output; each line  (where 1<=i<=10  ) contains the  of  in the form:  
N x i = result.*

**Sample Input:**

*2*

**Sample Output:**

*2 x 1 = 2*

*2 x 2 = 4*

*2 x 3 = 6*

*2 x 4 = 8*

*2 x 5 = 10*

*2 x 6 = 12*

*2 x 7 = 14*

*2 x 8 = 16*

*2 x 9 = 18*

*2 x 10 = 20*

**CODE:**

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

int N = Integer.parseInt(bufferedReader.readLine().trim());

for(int i=1; i<=10; i++)

{

int res=N\*i;

System.out.println(N+" x "+i+" = "+res);

}

bufferedReader.close();}

}

**JAVA LOOPS II**

***We use the integers , , and  to create the following series:***

***(a+2^0.b), (a+2^0.b+2^1.b),...,(a+2^0.b+2^1.b+...+2^n-1.b)***

***You are given q  queries in the form of a ,b , and n. For each query, print the series corresponding to the given  a, b, and n  values as a single line of  n space-separated integers*.**

**Input Format:**

***The first line contains an integer,q , denoting the number of queries.  
Each line i of the q  subsequent lines contains three space-separated integers describing the respective ai ,bi , and  ni values for that query.***

**Constraints:**

* ***0<=q<=500***
* ***0<=a,b<=50***
* ***1<=n<=15***

**Output Format:**

***For each query, print the corresponding series on a new line. Each series must be printed in order as a single line of  n space-separated integers.***

**Sample Input:**

***2***

***0 2 10***

***5 3 5***

**Sample Output:**

***2 6 14 30 62 126 254 510 1022 2046***

***8 14 26 50 98***

**Explanation:**

***We have two queries:***

1. ***We use a=0,b=2 , and n=10  to produce some series s0;s1,….sn-1:***
   * ***S0=0+1.2=2***
   * ***S1=0+1.2+2.2=6***
   * ***S2=0+1.2+2.2+4.2=14***

***... and so on.***

***Once we hit  n=10, we print the first ten terms as a single line of space-separated integers.***

1. ***We use a=5 ,b=3 , and n=5  to produce some series  s0;s1,….sn-1:***
   * ***S0=5+1.3=8***
   * ***S1=5+1.3+2.3=14***
   * ***S2=5+1.3+2.3+4.3=26***
   * ***S3=5+1.3+2.3+4.3+8.3=30***
   * ***S4=5+1.3+2.3+4.3+8.3+16.3=96***

***We then print each element of our series as a single line of space-separated values.***

**CODE:**

import java.util.\*;

import java.io.\*;

class Solution{

public static void main(String []argh){

Scanner in = new Scanner(System.in);

int t=in.nextInt();

for(int i=0;i<t;i++){

int a = in.nextInt();

int b = in.nextInt();

int n = in.nextInt();

for(int j=0;j<n;j++)

{

a=a+((int)Math.pow(2,j)\*b);

System.out.print(a+" ");

}

System.out.println();

}

in.close();

}

}

**JAVA DATATYPES**

***Java has 8 primitive data types;*char, boolean, byte, short, int, long, float, and double*. For this exercise, we'll work with the primitives used to hold integer values (*byte, short, int,*and*long*):***

* ***A*byte*is an 8-bit signed integer.***
* ***A*short*is a 16-bit signed integer.***
* ***An*int*is a 32-bit signed integer.***
* ***A*long*is a 64-bit signed integer.***

***Given an input integer , you must determine which primitive data types are capable of properly storing that input.***

***To get you started, a portion of the solution is provided for you in the editor.***

***Reference:***[***https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html***](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html)

**Input Format:**

***The first line contains an integer,T , denoting the number of test cases.  
Each test case,T , is comprised of a single line with an integer,n , which can be arbitrarily large or small.***

**Output Format:**

***For each input variable n  and appropriate primitive datatype , you must determine if the given primitives are capable of storing it. If yes, then print:***

***n can be fitted in:***

***\* dataType***

***If there is more than one appropriate data type, print each one on its own line and order them by size (i.e.: byte<short<int<long).***

***If the number cannot be stored in one of the four aforementioned primitives, print the line:***

***n can't be fitted anywhere.***

**Sample Input:**

***5***

***-150***

***150000***

***1500000000***

***213333333333333333333333333333333333***

***-100000000000000***

**Sample Output:**

***-150 can be fitted in:***

***\* short***

***\* int***

***\* long***

***150000 can be fitted in:***

***\* int***

***\* long***

***1500000000 can be fitted in:***

***\* int***

***\* long***

***213333333333333333333333333333333333 can't be fitted anywhere.***

***-100000000000000 can be fitted in:***

***\* long***

**Explanation:**

***-15  can be stored in a*short*, an*int*, or a*long*.***

***213333333333333333333333333333333333 is very large and is outside of the allowable range of values for the primitive data types discussed in this problem.***

**CODE:**

import java.util.\*;

import java.io.\*;

class Solution{

public static void main(String []argh)

{

Scanner sc = new Scanner(System.in);

int t=sc.nextInt();

for(int i=0;i<t;i++)

{

try

{

long x=sc.nextLong();

System.out.println(x+" can be fitted in:");

if(x>=Byte.MIN\_VALUE && x<=Byte.MAX\_VALUE)System.out.println("\* byte");

//Complete the code

if(x>=Short.MIN\_VALUE && x<=Short.MAX\_VALUE)System.out.println("\* short");

if(x>=Integer.MIN\_VALUE && x<=Integer.MAX\_VALUE)System.out.println("\* int");

if(x>=Long.MIN\_VALUE&& x<=Long.MAX\_VALUE);

System.out.println("\* long");

}

catch(Exception e)

{

System.out.println(sc.next()+" can't be fitted anywhere.");

}

}

}

}

**JAVA END-OF-FILE**

***The challenge here is to read n lines of input until you reach*EOF*, then number and print all n  lines of content.***

***Hint: Java's*Scanner.hasNext()*method is helpful for this problem.***

**Input Format:**

***Read some unknown n  lines of input from*stdin(System.in)*until you reach*EOF*; each line of input contains a non-empty*String*.***

**Output Format:**

***For each line, print the line number, followed by a single space, and then the line content received as input.***

**Sample Input:**

***Hello world***

***I am a file***

***Read me until end-of-file.***

**Sample Output:**

***1 Hello world***

***2 I am a file***

***3 Read me until end-of-file.***

**CODE:**

import java.io.\*;

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int i=1;

while(sc.hasNext())

{

System.out.println(i++ +" "+sc.nextLine());

}

}

}

**JAVA STATIC INITIALIZATION BLOCK**

***Static initialization blocks are executed when the class is loaded, and you can initialize static variables in those blocks.***

***It's time to test your knowledge of*Static initialization blocks*.***

***You are given a class*Solution*with a*main*method. Complete the given code so that it outputs the area of a parallelogram with breadth  B and height H. You should read the variables from the standard input.***

***If  B<=0 or H<=0  , the output should be*"java.lang.Exception: Breadth and height must be positive"*without quotes.***

**Input Format:**

***There are two lines of input. The first line contains B : the breadth of the parallelogram. The next line contains H : the height of the parallelogram.***

**Constraints:**

* ***100<=B<=100***
* ***100<=H<=100***

**Output Format:**

***If both values are greater than zero, then the*main*method must output the area of the*parallelogram*. Otherwise, print*"java.lang.Exception: Breadth and height must be positive"*without quotes.***

**Sample input 1:**

***1***

***3***

**Sample output 1:**

***3***

**Sample input 2:**

***-1***

***2***

**Sample output 2:**

***java.lang.Exception: Breadth and height must be positive***

**CODE:**

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

static int B,H;

static boolean flag=true;

static

{

Scanner sc=new Scanner(System.in);

B=sc.nextInt();

H=sc.nextInt();

try

{

if(B<=0 || H<=0)

{

throw new Exception("Breadth and height must be positive");

}

}

catch(Exception e)

{

System.out.println(e);

System.exit(0);

}

}

public static void main(String[] args){

        if(flag){

            int area=B\*H;

            System.out.print(area);

        }

    }//end of main

}//end of class

**JAVA INT TO STRING**

***You are given an integer n , you have to convert it into a string.***

***Please complete the partially completed code in the editor. If your code successfully converts  n into a string s  the code will print "*Good job*". Otherwise it will print "*Wrong answer*".***

***n can range between -100 to 100  inclusive.***

**Sample Input 0:**

***100***

**Sample Output 0:**

***Good job***

**CODE:**

import java.util.\*;

import java.security.\*;

public class Solution {

 public static void main(String[] args) {

  DoNotTerminate.forbidExit();

  try {

   Scanner in = new Scanner(System.in);

   int n = in .nextInt();

   in.close();

   String s=Integer.toString(n);

   if (n == Integer.parseInt(s)) {

    System.out.println("Good job");

   } else {

    System.out.println("Wrong answer.");

   }

  } catch (DoNotTerminate.ExitTrappedException e) {

   System.out.println("Unsuccessful Termination!!");

  }

 }

}

//The following class will prevent you from terminating the code using exit(0)!

class DoNotTerminate {

 public static class ExitTrappedException extends SecurityException {

  private static final long serialVersionUID = 1;

 }

 public static void forbidExit() {

  final SecurityManager securityManager = new SecurityManager() {

   @Override

   public void checkPermission(Permission permission) {

    if (permission.getName().contains("exitVM")) {

     throw new ExitTrappedException();

    }

   }

  };

  System.setSecurityManager(securityManager);

 }

}

**JAVA DATE AND TIME**

***The***[***Calendar class***](https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html)***is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY\_OF\_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week.***

***You are given a date. You just need to write the method, getday, which returns the*day*on that date. To simplify your task, we have provided a portion of the code in the editor.***

**Example:**

***Month=8***

***Day=14***

***Year=2017  
The method should return MONDAY  as the day on that date.***

******

***Function Description***

***Complete the*findDay*function in the editor below.***

**findDay*has the following parameters:***

* **int:*month***
* **int:*day***
* **int:*year***

***Returns***

* **string:*the day of the week in capital letters***

**Input Format:**

***A single line of input containing the space separated month, day and year, respectively, in MM DD YYYY   format.***

**Constraints:**

* ***2000<year<3000***

**Sample Input:**

***08 05 2015***

**Sample Output:**

***WEDNESDAY***

**Explanation:**

***The day on August 5 th 2015 was WEDNESDAY.***

**CODE:**

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.regex.\*;

 public static String findDay(int month, int day, int year) {

    String[] Day={"SUNDAY","MONDAY","TUESDAY","WEDNESDAY","THRUSDAY","FRIDAY","SATERDAY"};

        Calendar cal=Calendar.getInstance();

        cal.set(year,month-1,day);

        //System.out.print(cal.get(cal.DAY\_OF\_WEEK));

        int a=cal.get(Calendar.DAY\_OF\_WEEK) ;

    return  Day[a-1];

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        int month = Integer.parseInt(firstMultipleInput[0]);

        int day = Integer.parseInt(firstMultipleInput[1]);

        int year = Integer.parseInt(firstMultipleInput[2]);

        String res = Result.findDay(month, day, year);

        bufferedWriter.write(res);

        bufferedWriter.newLine();

        bufferedReader.close();

        bufferedWriter.close();

    }

}

**JAVA STRING INTRODUCTION**

***This exercise is to test your understanding of Java Strings. A sample String declaration:***

***String myString = "Hello World!"***

***The elements of a String are called characters. The number of characters in a String is called the length, and it can be retrieved with the String.length() method.***

***Given two strings of lowercase English letters,A  and B, perform the following operations:***

1. ***Sum the lengths of A  and B.***
2. ***Determine if A is lexicographically larger than  B(i.e.: does B  come before A in the dictionary?).***
3. ***Capitalize the first letter in  A and B  and print them on a single line, separated by a space.***

**Input Format:**

***The first line contains a string A . The second line contains another string B . The strings are comprised of only lowercase English letters.***

**Output Format:**

***There are three lines of output:  
For the first line, sum the lengths of  A and B.  
For the second line, write Yes if A is lexicographically greater than B  otherwise print No instead.  
For the third line, capitalize the first letter in both A and B  and print them on a single line, separated by a space.***

**Sample Input 0 :**

***hello***

***java***

**Sample Output 0:**

***9***

***No***

***Hello Java***

**Explanation 0:**

***String  A is "hello" and B is "java".***

***A  has a length of 5 , and B has a length of 4; the sum of their lengths is 9.  
When sorted alphabetically/lexicographically, "hello" precedes "java"; therefore, A  is not greater than B  and the answer is No.***

***When you capitalize the first letter of both A  and B  and then print them separated by a space, you get "Hello Java".***

**CODE:**

import java.io.\*;

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

String A=sc.next();

String B=sc.next();

int sum=(A.length()+B.length());

System.out.println(sum);

int n=A.compareTo(B);

if(n>0)

System.out.println("Yes");

else

System.out.println("No");

A = Character.toUpperCase(A.charAt(0)) + A.substring(1).toLowerCase();

B = Character.toUpperCase(B.charAt(0)) + B.substring(1).toLowerCase();

System.out.print(A + " " + B);

**JAVA SUBSTRING**

***Given a string,s , and two indices, start  and end, print a***[***substring***](https://en.wikipedia.org/wiki/Substring)***consisting of all characters in the inclusive range from start to end-1 . You'll find the*String*class'***[***substring method***](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#substring-int-int-)***helpful in completing this challenge.***

**Input Format:**

***The first line contains a single string denoting s.  
The second line contains two space-separated integers denoting the respective values of  start and end .***

**Constraints:**

* ***1<=|s|<=100***
* ***0<=start<end<=n***
* ***String s  consists of English alphabetic letters (i.e.,[a-z A-Z] ) only.***

**Output Format:**

***Print the substring in the inclusive range from start  to end-1 .***

**Sample Input:**

***Helloworld***

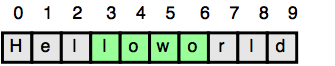
***3 7***

**Sample Output:**

***Lowo***

**Explanation:**

***In the diagram below, the substring is highlighted in*green*:***

******

**CODE:**

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

String S = in.next();

int start = in.nextInt();

int end = in.nextInt();

System.out.println(S.substring(start,end));

}

}

**JAVA SUBSTRING COMPARISIONS**

***We define the following terms:***

* [***Lexicographical Order***](https://en.wikipedia.org/wiki/Lexicographical_order)***, also known as*alphabetic*or*dictionary*order, orders characters as follows:***
* ***A<B<…….<Y<Z<a<b<……..<y<z***

***For example, ball < cat, dog < dorm, Happy < happy, Zoo < ball.***

* ***A***[***substring***](https://en.wikipedia.org/wiki/Substring)***of a string is a contiguous block of characters in the string. For example, the substrings of abc are a, b, c, ab, bc, and abc.***

***Given a string,s , and an integer,k , complete the function so that it finds the lexicographically*smallest*and*largest*substrings of length  k.***

**Function Description:**

***Complete the*getSmallestAndLargest*function in the editor below.***

**getSmallestAndLargest*has the following parameters:***

* **string s:*a string***
* **int k:*the length of the substrings to find***

**Returns:**

* **string:*the string ' + "\n" + ' where and are the two substrings***

**Input Format:**

***The first line contains a string denoting s.  
The second line contains an integer denoting k.***

**Constraints:**

* ***1<=|s|<=1000***
* ***s consists of English alphabetic letters only (i.e., [a-zA-Z]).***

**Sample Input 0:**

***welcometojava***

***3***

**Sample Output 0:**

***ava***

***wel***

**Explanation 0:**

***String s=”welcometojava”  has the following lexicographically-ordered substrings of length k=3 :***

***[“ava”,”com”,”ele”,”eto”,”jav”,”lco”,”met”,”oja”…]***

***We then return the first (lexicographically smallest) substring and the last (lexicographically largest) substring as two newline-separated values (i.e., ava\nwel).***

***The stub code in the editor then prints ava as our first line of output and wel as our second line of output.***

**CODE:**

import java.util.Scanner;

public class Solution{

public static String getSmallestAndLargest(String s, int k) {

String str=s.substring(0,k);

String smallest = str;

String largest = str;

for(int i=1;i<=s.length()-k;i++)

{

str=s.substring(i,i+k);

if(str.compareTo(smallest)<0)

smallest=str;

if(str.compareTo(largest)>0)

largest=str;

}

/\*String smallest = "";

String largest = "";

SortedSet<String> S1=new TreeSet<String>();

for(int i=1;i<=s.length()-k;i++)

{

S1.add(s.substring(i,i+k));

}

smallest=S1.first();

largest=S1.last();\*/

return smallest + "\n" + largest;

}

 public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String s = scan.next();

        int k = scan.nextInt();

        scan.close();

        System.out.println(getSmallestAndLargest(s, k));

    }

}

**JAVA STRING REVERSE**

***A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward.***

***Given a string A , print Yes if it is a palindrome, print No otherwise.***

***Constraints***

* ***A  will consist at most  lower case 50 english letters.***

***Sample Input***

***madam***

***Sample Output***

***Yes***

**CODE:**

import java.io.\*;

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

String A=sc.next();

String rev="";

/\* Enter your code here. Print output to STDOUT. \*/

for(int i=A.length()-1;i>=0;i--)

{

rev=rev+A.charAt(i);

}

if(A.equals(rev))

System.out.println("Yes");

else

System.out.println("No");

}

}

**JAVA ANAGRAMS**

***Two strings, a  and b, are called anagrams if they contain all the same characters in the same frequencies. For this challenge, the test is not case-sensitive. For example, the anagrams of CAT are CAT, ACT, tac, TCA, aTC, and CtA.***

***Function Description***

***Complete the isAnagram function in the editor.***

***isAnagram has the following parameters:***

* ***string a: the first string***
* ***string b: the second string***

**Returns:**

* ***boolean: If  a and  are b case-insensitive anagrams, return true. Otherwise, return false.***

**Input Format:**

***The first line contains a string a .  
The second line contains a string b .***

**Constraints:**

* ***1<=length(a),length(b)<=50***
* ***Strings a  and b consist of English alphabetic characters.***
* ***The comparison should NOT be case sensitive.***

**Sample Input 0:**

***anagram***

***margana***

**Sample Output 0:**

***Anagrams***

**Explanation 0:**

| ***Character*** | ***Frequency: anagram*** | ***Frequency: margana*** |
| --- | --- | --- |
| ***A or a*** | ***3*** | ***3*** |
| ***G or g*** | ***1*** | ***1*** |
| ***N or n*** | ***1*** | ***1*** |
| ***M or m*** | ***1*** | ***1*** |
| ***R or r*** | ***1*** | ***1*** |

***The two strings contain all the same letters in the same frequencies, so we print "Anagrams".***

**Sample Input 1:**

***anagramm***

***marganaa***

**Sample Output 1:**

***Not Anagrams***

**Explanation 1:**

| ***Character*** | ***Frequency: anagramm*** | ***Frequency: marganaa*** |
| --- | --- | --- |
| ***A or a*** | ***3*** | ***4*** |
| ***G or g*** | ***1*** | ***1*** |
| ***N or n*** | ***1*** | ***1*** |
| ***M or m*** | ***2*** | ***1*** |
| ***R or r*** | ***1*** | ***1*** |

***The two strings don't contain the same number of a's and m's, so we print "Not Anagrams".***

**Sample Input 2:**

***Hello***

***hello***

**Sample Output 2:**

***Anagrams***

**Explanation 2:**

| ***Character*** | ***Frequency: Hello*** | ***Frequency: hello*** |
| --- | --- | --- |
| ***E or e*** | ***1*** | ***1*** |
| ***H or h*** | ***1*** | ***1*** |
| ***L or l*** | ***2*** | ***2*** |
| ***O or o*** | ***1*** | ***1*** |

***The two strings contain all the same letters in the same frequencies, so we print "Anagrams".***

**CODE:**

import java.util.Scanner;

public class Solution {

static boolean isAnagram(String a, String b) {

a = a.toLowerCase();

b = b.toLowerCase();

if(a.length()==b.length())

{

char[] Cs1 = a.toCharArray();

char[] Cs2 = b.toCharArray();

java.util.Arrays.sort(Cs1);

java.util.Arrays.sort(Cs2);

if(java.util.Arrays.equals(Cs1, Cs2))

return true;

}

return false;

}

 public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String a = scan.next();

        String b = scan.next();

        scan.close();

        boolean ret = isAnagram(a, b);

        System.out.println( (ret) ? "Anagrams" : "Not Anagrams" );

    }

}

**JAVA USERNAME REGULAR EXPRESSION**

***You are updating the username policy on your company's internal networking platform. According to the policy, a username is considered valid if all the following constraints are satisfied:***

* ***The username consists of  8  to 30 characters inclusive. If the username consists of less than  8 or greater than 30 characters, then it is an invalid username.***
* ***The username can only contain alphanumeric characters and underscores (\_). Alphanumeric characters describe the character set consisting of lowercase characters [a-z], uppercase characters[A-Z] , and digits [0-9].***
* ***The first character of the username must be an alphabetic character, i.e., either lowercase character [a-z]  or uppercase character [A-Z] .***

**For example:**

| ***Username*** | ***Validity*** |
| --- | --- |
| ***Julia*** | ***INVALID; Username length < 8 characters*** |
| ***Samantha*** | ***VALID*** |
| ***Samantha\_21*** | ***VALID*** |
| ***1Samantha*** | ***INVALID; Username begins with non-alphabetic character*** |
| ***Samantha?10\_2A*** | ***INVALID; '?' character not allowed*** |

***Update the value of regularExpression field in the UsernameValidator class so that the regular expression only matches with valid usernames.***

**Input Format:**

***The first line of input contains an integer , describing the total number of usernames. Each of the next  lines contains a string describing the username. The locked stub code reads the inputs and validates the username.***

**Constraints:**

* ***1<=n<=100***
* ***The username consists of any printable characters.***

**Output Format:**

***For each of the usernames, the locked stub code prints Valid if the username is valid; otherwise Invalid each on a new line.***

**Sample Input 0:**

***8***

***Julia***

***Samantha***

***Samantha\_21***

***1Samantha***

***Samantha?10\_2A***

***JuliaZ007***

***Julia@007***

***\_Julia007***

**Sample Output 0:**

***Invalid***

***Valid***

***Valid***

***Invalid***

***Invalid***

***Valid***

***Invalid***

***Invalid***

**Explanation 0:**

***Refer diagram in the challenge statement.***

**CODE:**

import java.util.Scanner;

class UsernameValidator {

public static final String regularExpression = "^[a-zA-Z]\\w{7,29}$";

}

public class Solution {

    private static final Scanner scan = new Scanner(System.in);

    public static void main(String[] args) {

        int n = Integer.parseInt(scan.nextLine());

        while (n-- != 0) {

            String userName = scan.nextLine();

            if (userName.matches(UsernameValidator.regularExpression))   {

                System.out.println("Valid");

            }

else

{

                System.out.println("Invalid");

            }

        }

    }

}

**JAVA PRIMALITY TEST**

***A prime number is a natural number greater than 1 whose only positive divisors are 1  and itself. For example, the first six prime numbers are 2,3,5,7,11 and 13.***

***Given a large integer, n, use the Java BigInteger class' [isProbablePrime](https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html" \l "isProbablePrime%28int%29) method to determine and print whether it's prime or not prime.***

**Input Format:**

***A single line containing an integer, n (the number to be checked).***

**Constraints:**

* ***n contains at most 100  digits.***

**Output Format:**

***If n  is a prime number, print prime; otherwise, print not prime.***

**Sample Input:**

***13***

**Sample Output:**

***prime***

**Explanation:**

***The only positive divisors of 13  are 1  and 13, so we print prime.***

**CODE:**

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String[] args) throws IOException {

//BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

Scanner sc=new Scanner(System.in);

// String n = bufferedReader.readLine();

BigInteger n=new BigInteger(sc.next());

System.out.println(n.isProbablePrime(20)?"prime":"not prime");

sc.close();

}

}

**JAVA BIGINTEGER**

***In this problem, you have to add and multiply huge numbers! These numbers are so big that you can't contain them in any ordinary data types like a long integer.***

***Use the power of Java's BigInteger class and solve this problem.***

**Input Format:**

***There will be two lines containing two numbers, a and b.***

**Constraints:**

***a  and b  are non-negative integers and can have maximum  200 digits.***

**Output Format:**

***Output two lines. The first line should contain a+b , and the second line should contain axb . Don't print any leading zeros.***

**Sample Input:**

***1234***

***20***

**Sample Output:**

***1254***

***24680***

***Explanation***

***1234+20=1254***

***1234\*20=24680***

**CODE:**

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

    public static void main(String[] args) {

       Scanner sc=new Scanner(System.in);

       String a=sc.nextLine();

       String b=sc.nextLine();

        BigInteger x= new BigInteger(a);

         BigInteger y= new BigInteger(b);

         System.out.println(x.add(y));

        System.out.println(x.multiply(y));

        sc.close();

    }

}

**JAVA 1D ARRAY**

***An array is a simple data structure used to store a collection of data in a contiguous block of memory. Each element in the collection is accessed using an index, and the elements are easy to find because they're stored sequentially in memory.***

***Because the collection of elements in an array is stored as a big block of data, we typically use arrays when we know exactly how many pieces of data we're going to have. For example, you might use an array to store a list of student ID numbers, or the names of state capitals. To create an array of integers named myArray  that can hold four integer values, you would write the following code:***

***int[] myArray = new int[4];***

***This sets aside a block of memory that's capable of storing 4 integers. Each integer storage cell is assigned a unique index ranging from  0 to one less than the size of the array, and each cell initially contains a 0 . In the case of myArray , we can store integers at indices 0 ,1 ,2 , and 3. Let's say we wanted the last cell to store the number 12 ; to do this, we write:***

***myArray[3] = 12;***

***Similarly, we can print the contents of the last cell with the following code:***

***System.out.println(myArray[3]);***

***The code above prints the value stored at index 3 of myArray , which is 12  (the value we previously stored there). It's important to note that while Java initializes each cell of an array of integers with a , not all languages do this.***

**Task:**

***The code in your editor does the following:***

1. ***Reads an integer from stdin and saves it to a variable,n , denoting some number of integers.***
2. ***Reads n  integers corresponding to a0,a1,….,an-1  from stdin and saves each integer ai  to a variable,val, .***
3. ***Attempts to print each element of an array of integers named a .***

***Write the following code in the unlocked portion of your editor:***

1. ***Create an array,a , capable of holding n  integers.***
2. ***Modify the code in the loop so that it saves each sequential value to its corresponding location in the array. For example, the first value must be stored in  a0, the second value must be stored in a1, and so on.***

***Good luck!***

**Input Format:**

***The first line contains a single integer,n , denoting the size of the array.  
Each line i of the n  subsequent lines contains a single integer denoting the value of element ai .***

**Output Format:**

***You are not responsible for printing any output to stdout. Locked code in the editor loops through array a  and prints each sequential element on a new line*.**

**Sample Input:**

***5***

***10***

***20***

***30***

***40***

***50***

**Sample Output:**

***10***

***20***

***30***

***40***

***50***

**Explanation:**

***When we save each integer to its corresponding index in a , we get a=[10,20,30,40,50] . The locked code prints each array element on a new line from left to right.***

**CODE:**

import java.util.\*;

public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n=scan.nextInt();

int[] a=new int[n];

for(int i=0;i<n;i++)

{

    a[i]=scan.nextInt();

}

        scan.close();

        // Prints each sequential element in array a

        for (int i = 0; i < a.length; i++) {

            System.out.println(a[i]);

        }

    }

}

**JAVA ARRAYLIST**

***Sometimes it's better to use dynamic size arrays. Java's [Arraylist](https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) can provide you this feature. Try to solve this problem using Arraylist.***

***You are given n lines. In each line there are zero or more integers. You need to answer a few queries where you need to tell the number located in yth  position of xth line.***

***Take your input from System.in.***

**Input Format: *The first line has an integer. In each of the next n lines there will be an integer d denoting number of integers on that line and then there will be d  space-separated integers. In the next line there will be an integer q  denoting number of queries. Each query will consist of two integers  x and y .***

**Constraints:**

* ***1<=n<=20000***
* ***0<=d<=50000***
* ***1<=q<=1000***
* ***1<=x<=n***

***Each number will fit in signed integer.  
Total number of integers in  n lines will not cross 105 .***

**Output Format: *In each line, output the number located in yth  position of xth  line. If there is no such position, just print "ERROR!"***

**Sample Input:**

***5***

***5 41 77 74 22 44***

***1 12***

***4 37 34 36 52***

***0***

***3 20 22 33***

***5***

***1 3***

***3 4***

***3 1***

***4 3***

***5 5***

**Sample Output:**

***74***

***52***

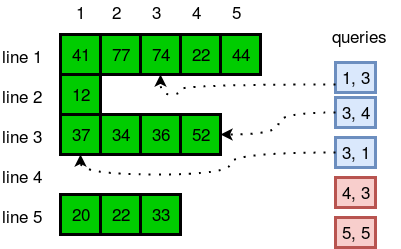
***37***

***ERROR!***

***ERROR!***

**Explanation:**

***The diagram below explains the queries:***

******

**CODE:**

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String[] args) {

ArrayList[] ar=new ArrayList[20000];

Scanner sc =new Scanner(System.in);

int n=sc.nextInt();

for(int i=0;i<n;i++)

{

ar[i]=new ArrayList();

int d=sc.nextInt();

for(int j=0;j<d;j++)

{

int value=sc.nextInt();

ar[i].add(value);

}

} int query=sc.nextInt();

for(int i=1;i<=query;i++)

{

int row=sc.nextInt();

int col=sc.nextInt();

try{

System.out.println(ar[row-1].get(col-1));

}

catch(Exception e)

{

System.out.println("ERROR!");

}

}

}

}

**JAVA LIST**

***For this problem, we have 2 types of queries you can perform on a***[***List***](https://docs.oracle.com/javase/7/docs/api/java/util/List.html)***:***

1. ***Insert y at index x :***
2. ***Insert***

***x y***

1. ***Delete the element at index x :***
2. ***Delete***

***x***

***Given a list, L, of N integers, perform Q queries on the list. Once all queries are completed, print the modified list as a single line of space-separated integers*.**

**Input Format:**

***The first line contains an integer, N (the initial number of elements in L ).  
The second line contains N space-separated integers describing L .  
The third line contains an integer,Q  (the number of queries).  
The 2Q  subsequent lines describe the queries, and each query is described over two lines:***

* ***If the first line of a query contains the String Insert, then the second line contains two space separated integers x,y , and the value y  must be inserted into L  at index x .***
* ***If the first line of a query contains the String Delete, then the second line contains index x , whose element must be deleted from L .***

**Constraints:**

* ***1<=N<=4000***
* ***1<=Q<=4000***
* ***Each element in is a*32-bit integer*.***

**Output Format:**

***Print the updated list L  as a single line of space-separated integers.***

**Sample Input:**

***5***

***12 0 1 78 12***

***2***

***Insert***

***5 23***

***Delete***

***0***

***Sample Output***

***0 1 78 12 23***

**Explanation:**

***L=[12,0,1,78,12]***

***Q1= Insert 23 at index 5 .***

***L0=[12,0,1,78,12,23]  
Q1= Delete the element at index 0 .  
L1=[0,1,78,12,23]***

***Having performed all Q  queries, we print L1  as a single line of space-separated integers.***

**CODE:**

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

    public static void main(String[] args) {

        /\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

        Scanner sc=new Scanner(System.in);

        int n=sc.nextInt();

        ArrayList<Integer> ar=new ArrayList<>();

        for(int i=0; i<n; i++)

        {

            int element=sc.nextInt();

            ar.add(element);

        }

        int q=sc.nextInt();

        for(int i=0; i<q; i++)

        {

             String query=sc.next();

            if(query.equals("Insert")){

            int x=sc.nextInt();

            int y=sc.nextInt();

            ar.add(x,y);

            }

            else{

                int z=sc.nextInt();

                ar.remove(z);

            }

        }

        /\*for(int i=0; i<ar.size(); i++)

        {

            System.out.println(ar);

        }\*/

        Iterator<Integer> it=ar.iterator();

        while(it.hasNext())

        {

            System.out.print(it.next()+" ");

        }

    }

}

**JAVA STACK**

***In computer science, a stack or LIFO (last in, first out) is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the last element that was added.(Wikipedia)***

***A string containing only parentheses is balanced if the following is true: 1. if it is an empty string 2. if A and B are correct, AB is correct, 3. if A is correct, (A) and {A} and [A] are also correct.***

***Examples of some correctly balanced strings are: "{}()", "[{()}]", "({()})"***

***Examples of some unbalanced strings are: "{}(", "({)}", "[[", "}{" etc.***

***Given a string, determine if it is balanced or not.***

**Input Format:**

***There will be multiple lines in the input file, each having a single non-empty string. You should read input till end-of-file.***

***The part of the code that handles input operation is already provided in the editor.***

**Output Format:**

***For each case, print 'true' if the string is balanced, 'false' otherwise.***

**Sample Input:**

***{}()***

***({()})***

***{}(***

***[]***

**Sample Output:**

***true***

***true***

***false***

***true***

**CODE:**

import java.util.\*;

class Solution{

public static void main(String []argh)

{

Scanner sc = new Scanner(System.in);

while (sc.hasNext()) {

String input=sc.next();

System.out.println(isvalid(input));

}

}

public static boolean isvalid(String s)

{

Stack<Character> st=new Stack<Character>();

int length=s.length();

int i;

for( i=0;i<length;i++)

{

char ch=s.charAt(i);

if(ch=='('|| ch=='{'|| ch=='[')

st.push(ch);

else if(st.isEmpty())

return false;

else if(st.peek()=='(' && ch==')')

st.pop();

else if(st.peek()=='{' && ch=='}')

st.pop();

else if(st.peek()=='[' && ch==']')

st.pop();

else

break;

}

if(i==length && st.isEmpty())

return true;

else

return false;

}

}

**JAVA STACK**

***In computer science, a stack or LIFO (last in, first out) is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the last element that was added.(Wikipedia)***

***A string containing only parentheses is balanced if the following is true: 1. if it is an empty string 2. if A and B are correct, AB is correct, 3. if A is correct, (A) and {A} and [A] are also correct.***

***Examples of some correctly balanced strings are: "{}()", "[{()}]", "({()})"***

***Examples of some unbalanced strings are: "{}(", "({)}", "[[", "}{" etc.***

***Given a string, determine if it is balanced or not.***

**Input Format:**

***There will be multiple lines in the input file, each having a single non-empty string. You should read input till end-of-file.***

***The part of the code that handles input operation is already provided in the editor.***

**Output Format:**

***For each case, print 'true' if the string is balanced, 'false' otherwise.***

**Sample Input:**

***{}()***

***({()})***

***{}(***

***[]***

**Sample Output:**

***true***

***true***

***false***

***true***

**CODE:**

import java.util.\*;

class Solution{

public static void main(String []argh)

{

Scanner sc = new Scanner(System.in);

while (sc.hasNext()) {

String input=sc.next();

System.out.println(isvalid(input));

}

}

public static boolean isvalid(String s)

{

Stack<Character> st=new Stack<Character>();

int length=s.length();

int i;

for( i=0;i<length;i++)

{

char ch=s.charAt(i);

if(ch=='('|| ch=='{'|| ch=='[')

st.push(ch);

else if(st.isEmpty())

return false;

else if(st.peek()=='(' && ch==')')

st.pop();

else if(st.peek()=='{' && ch=='}')

st.pop();

else if(st.peek()=='[' && ch==']')

st.pop();

else

break;

}

if(i==length && st.isEmpty())

return true;

else

return false

}

}

**JAVA HASHSET**

***In computer science, a set is an abstract data type that can store certain values, without any particular order, and no repeated values(Wikipedia). {1,2,3} is an example of a set, but {1,2,2} is not a set. Today you will learn how to use sets in java by solving this problem.***

***You are given n  pairs of strings. Two pairs (a,b) and (c,d)  are identical if a=c  and b=d . That also implies (a,b)  is not same as (b,a) . After taking each pair as input, you need to print number of unique pairs you currently have.***

***Complete the code in the editor to solve this problem.***

**Input Format:**

***In the first line, there will be an integer T  denoting number of pairs. Each of the next  T lines will contain two strings seperated by a single space.***

**Constraints:**

* ***1<=T<=100000***
* ***Length of each string is atmost 5  and will consist lower case letters only.***

**Output Format:**

***Print  T lines. In the ith line, print number of unique pairs you have after taking  ith pair as input.***

**Sample Input:**

***5***

***john tom***

***john mary***

***john tom***

***mary anna***

***mary anna***

**Sample Output:**

***1***

***2***

***2***

***3***

***3***

**Explanation:**

* ***After taking the first input, you have only one pair: (john,tom)***
* ***After taking the second input, you have two pairs: (john, tom) and (john, mary)***
* ***After taking the third input, you still have two unique pairs.***
* ***After taking the fourth input, you have three unique pairs: (john,tom), (john, mary) and (mary, anna)***
* ***After taking the fifth input, you still have three unique pairs.***

**CODE:**

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

 public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        int t = s.nextInt();

        String [] pair\_left = new String[t];

        String [] pair\_right = new String[t];

        for (int i = 0; i < t; i++) {

            pair\_left[i] = s.next();

            pair\_right[i] = s.next();

        }

HashSet<String> hs=new HashSet<>();

for(int i=0; i<t; i++)

{

  hs.add(pair\_left[i]+" "+pair\_right[i]);

  System.out.println(hs.size());

}

    }

}

**JAVA SORT**

***You are given a list of student information: ID, FirstName, and CGPA. Your task is to rearrange them according to their CGPA in decreasing order. If two student have the same CGPA, then arrange them according to their first name in alphabetical order. If those two students also have the same first name, then order them according to their ID. No two students have the same ID.***

***Hint: You can use comparators to sort a list of objects. See the***[***oracle docs***](http://docs.oracle.com/javase/tutorial/collections/interfaces/order.html)***to learn about comparators.***

**Input Format:**

***The first line of input contains an integer N , representing the total number of students. The next N  lines contains a list of student information in the following structure:***

***ID Name CGPA***

**Constraints:**

***3<=N<=1000***

***0<=ID<=100000***

***5<=|Name|<=30***

***0<=CGPA<=4.00***

***The name contains only lowercase English letters. The ID  contains only integer numbers without leading zeros. The*CGPA*will contain, at most, 2 digits after the decimal point.***

**Output Format:**

***After rearranging the students according to the above rules, print the first name of each student on a separate line.***

**Sample Input:**

***5***

***33 Rumpa 3.68***

***85 Ashis 3.85***

***56 Samiha 3.75***

***19 Samara 3.75***

***22 Fahim 3.76***

**Sample Output:**

***Ashis***

***Fahim***

***Samara***

***Samiha***

***Rumpa***

**CODE:**

import java.util.\*;

class Student{

private int id;

private String fname;

private double cgpa;

public Student(int id, String fname, double cgpa) {

super();

this.id = id;

this.fname = fname;

this.cgpa = cgpa;

}

public int getId() {

return id;

}

public String getFname() {

return fname;

}

public double getCgpa() {

return cgpa;

}

}

//Complete the code

public class Solution

{

public static void main(String[] args){

Scanner in = new Scanner(System.in);

int testCases = Integer.parseInt(in.nextLine());

List<Student> studentList = new ArrayList<Student>();

while(testCases>0){

int id = in.nextInt();

String fname = in.next();

double cgpa = in.nextDouble();

Student st = new Student(id, fname, cgpa);

studentList.add(st);

testCases--;

}

Collections.sort(studentList,Comparator.comparing(Student::getCgpa).reversed().thenComparing(Student::getFname).thenComparing(Student::getId));

for(Student st: studentList){

System.out.println(st.getFname());

}

}

}

**JAVA ITERATOR**

***Java Iterator class can help you to iterate through every element in a collection.* Here is a simple example:**

***import java.util.\*;***

***public class Example{***

***public static void main(String []args){***

***ArrayList mylist = new ArrayList();***

***mylist.add("Hello");***

***mylist.add("Java");***

***mylist.add("4");***

***Iterator it = mylist.iterator();***

***while(it.hasNext()){***

***Object element = it.next();***

***System.out.println((String)element);***

***}***

***}***

***}***

***In this problem you need to complete a method*func*. The method takes an*ArrayList*as input. In that*ArrayList*there is one or more integer numbers, then there is a special string "###", after that there are one or more other strings. A* sample ArrayList may look like this:**

***element[0]=>42***

***element[1]=>10***

***element[2]=>"###"***

***element[3]=>"Hello"***

***element[4]=>"Java"***

***You have to modify the*func*method by editing at most 2 lines so that the code only prints the elements after the special string "###". For the sample above the* output will be:**

***Hello***

***Java***

**Note:*The stdin doesn't contain the string*"###"*, it is added in the*main*method.***

***To restore the original code in the editor, click the top left icon on the editor and create a new buffer.***

**CODE:**

import java.util.\*;

public class Main{

   static Iterator func(ArrayList mylist){

      Iterator it=mylist.iterator();

      while(it.hasNext()){

         Object element = it.next();

if(element.equals("###"))//Hints: use instanceof operator

            break;

        }

      return it;

   }

   @SuppressWarnings({ "unchecked" })

   public static void main(String []args){

      ArrayList mylist = new ArrayList();

      Scanner sc = new Scanner(System.in);

      int n = sc.nextInt();

      int m = sc.nextInt();

      for(int i = 0;i<n;i++){

         mylist.add(sc.nextInt());

      }

      mylist.add("###");

      for(int i=0;i<m;i++){

         mylist.add(sc.next());

      }

      Iterator it=func(mylist);

      while(it.hasNext()){

         Object element = it.next();

         System.out.println((String)element);

      }

   }

}