\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

From PrepInsta’s top 25 HackerRank questions , here are a few examples:

1. **Maximum Passengers Problem** – Matrix traversal with constraints.
2. **Minimum Street Lights** – Greedy algorithm for coverage.
3. **Maximize Earnings** – Job scheduling with profit optimization.
4. **Duplicate characters in a string**
5. **Second largest number in an array**
6. **Anagram check**
7. **Armstrong number validation**

Leetcode:

|  |
| --- |
| FIZZBUZZ  import java.util.ArrayList;  public class Exercise {  public String[] fizzBuzz(int n) {  ArrayList<String> arrayList = new ArrayList<>();  for (int i = 1; i <= n; i++) {  if (i % 3 == 0 && i % 5 == 0) {  arrayList.add("FizzBuzz");  } else if (i % 3 == 0) {  arrayList.add("Fizz");  } else if (i % 5 == 0) {  arrayList.add("Buzz");  } else {  **arrayList.add(String.valueOf(i));**  }  }  **return arrayList.toArray(new String[0]);**  }  } |

|  |
| --- |
|  |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Most HackerRank challenges require you to read input from [stdin](https://en.wikipedia.org/wiki/Standard_streams#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);

**Why the Fix?**

* nextInt() and nextDouble() do **not consume the newline** (\n) after the number.
* nextLine() reads until the end of the line, so if you don’t consume the leftover newline, it will read an empty string.
* Adding sc.nextLine() before reading the actual string input ensures the buffer is clean.
* import java.io.\*;
* import java.util.\*;
* 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 c=sc.nextInt();
* Double b=sc.nextDouble();
* sc.nextLine();
* String a=sc.nextLine();

* System.out.println("String: "+a.toString());
* System.out.println("Double: "+b);
* System.out.println("Int: "+c);
* }
* }

A screenshot of a computer

AI-generated content may be incorrect.

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

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++){

             String a=N+" x "+ i+" = ";

             int b=N\*i;

            System.out.println(a+b);

         }

        bufferedReader.close();

    }

}

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**import java.io.\*;**

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

**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 q = sc.nextInt(); // number of queries**

**for (int i = 0; i < q; i++) {**

**int a = sc.nextInt();**

**int b = sc.nextInt();**

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

**int sum = a;**

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

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

**System.out.print(sum + " ");**

**}**

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

**}**

**sc.close();**

**}**

**}**

**✅ Problem Summary**

**You are given q queries. Each query contains three integers: a, b, and n. For each query, you need to print a series of n space-separated integers using the formula:**

**Si=a+b×(20+21+⋯+2i)*Si*​=*a*+*b*×(20+21+⋯+2*i*)**

**📥 Input Format**

**The first line contains an integer q (number of queries).**

**Each of the next q lines contains three space-separated integers: a, b, n.**

**📤 Output Format**

**For each query, print the series on a new line.**

**💡 Java Code Solution**

**🧪 Example Input**

**2**

**0 2 10**

**5 3 5**

**✅ Example Output**

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

**8 14 26 50 98**

**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.**

**Input Format**

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

**Output Format**

**For each input variable  and appropriate primitive , 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.: ).**

**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**

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

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

**A screenshot of a computer

AI-generated content may be incorrect.**

**import java.io.\*;**

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

**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 t = sc.nextInt(); // number of test cases**

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

**try {**

**long n = sc.nextLong();**

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

**if (n >= Byte.MIN\_VALUE && n <= Byte.MAX\_VALUE)**

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

**if (n >= Short.MIN\_VALUE && n <= Short.MAX\_VALUE)**

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

**if (n >= Integer.MIN\_VALUE && n <= Integer.MAX\_VALUE)**

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

**if (n >= Long.MIN\_VALUE && n <= Long.MAX\_VALUE)**

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

**} catch (Exception e) {**

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

**}**

**}**

**sc.close();**

**}**

**}**

**Here’s the complete Java solution for the problem you described, which checks whether a given number can fit into standard Java primitive data types: byte, short, int, and long.**

**✅ Java Code**

**📥 Input Format**

**5**

**-150**

**150000**

**1500000000**

**213333333333333333333333333333333333**

**-100000000000000**

**📤 Output Format**

**-150 can be fitted in:**

**\* short333333333333333333333333333333333 can't be fitted anywhere.**

**-100000000000000 can be fitted in:**

**\* long**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**"In computing, *End Of File* (commonly abbreviated *EOF*) is a condition in a computer operating system where no more data can be read from a data source." — (**[**Wikipedia: End-of-file**](https://en.wikipedia.org/wiki/End-of-file)**)**

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

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

**Input Format**

**Read some unknown  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.**

**Here’s the complete Java solution for reading input until EOF (End of File) and printing each line with its line number:**

**✅ Java Code Using Scanner.hasNext()**

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

**public class Solution {**

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

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

**int lineNumber = 1;**

**while (sc.hasNext()) {**

**String line = sc.nextLine();**

**System.out.println(lineNumber + " " + line);**

**lineNumber++;**

**}**

**sc.close();**

**}**

**}**

**📥 Input Format**

**You don’t know how many lines will be provided. Each line contains a non-empty string.**

**📤 Output Format**

**For each line, print:**

**<line\_number> <line\_content>**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

[**Initializing Fields (The Java™ Tutorials > Learning the Java Language > Classes and Objects)**](https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html)

**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, , which returns the *day* on that date. To simplify your task, we have provided a portion of the code in the editor.**

**Example**

**The method should return  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    format.**

**Constraints**

**Sample Input**

**08 05 2015**

**Sample Output**

**WEDNESDAY**

**Explanation**

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

**import java.io.\*;**

**import java.math.\*;**

**import java.security.\*;**

**import java.text.\*;**

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

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

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

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

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

**import static java.util.stream.Collectors.joining;**

**import static java.util.stream.Collectors.toList;**

**class Result {**

**/\***

**\* Complete the 'findDay' function below.**

**\***

**\* The function is expected to return a STRING.**

**\* The function accepts following parameters:**

**\*  1. INTEGER month**

**\*  2. INTEGER day**

**\*  3. INTEGER year**

**\*/**

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

**Calendar cal = Calendar.getInstance();**

**cal.set(Calendar.YEAR, year);**

**cal.set(Calendar.MONTH, month - 1); // Months are 0-based in Calendar**

**cal.set(Calendar.DAY\_OF\_MONTH, day);**

**String[] days = { "SUNDAY", "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY", "SATURDAY" };**

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

**return days[dayOfWeek - 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();**

**}**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Given a double-precision number, , denoting an amount of money, use the NumberFormat class' getCurrencyInstance method to convert into the US, Indian, Chinese, and French currency formats. Then print the formatted values as follows: US: formattedPayment India: formattedPayment China: formattedPayment France: formattedPayment where is formatted according to the appropriate Locale's currency. Note: India does not have a built-in Locale, so you must construct one where the language is en (i.e., English). Input Format A single double-precision number denoting . Constraints Output Format On the first line, print US: u where is formatted for US currency. On the second line, print India: i where is formatted for Indian currency. On the third line, print China: c where is formatted for Chinese currency. On the fourth line, print France: f, where is formatted for French currency. Sample Input 12324.134 Sample Output US: $12,324.13 India: Rs.12,324.13 China: ￥12,324.13 France: 12 324,13 € Explanation Each line contains the value of formatted according to the four countries' respective currencies.**

**Copilot said:**

**Copilot**

**Here’s the complete Java solution using NumberFormat.getCurrencyInstance() to format a given double value into US, India, China, and France currency formats:**

**✅ Java Code**

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

**import java.text.\*;**

**public class Solution {**

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

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

**double payment = sc.nextDouble();**

**sc.close();**

**// US Locale**

**Locale us = Locale.US;**

**// India Locale (custom, since Java doesn't have a built-in one)**

**Locale india = new Locale("en", "IN");**

**// China Locale**

**Locale china = Locale.CHINA;**

**// France Locale**

**Locale france = Locale.FRANCE;**

**// Format currency**

**NumberFormat usFormat = NumberFormat.getCurrencyInstance(us);**

**NumberFormat indiaFormat = NumberFormat.getCurrencyInstance(india);**

**NumberFormat chinaFormat = NumberFormat.getCurrencyInstance(china);**

**NumberFormat franceFormat = NumberFormat.getCurrencyInstance(france);**

**// Output**

**System.out.println("US: " + usFormat.format(payment));**

**System.out.println("India: " + indiaFormat.format(payment));**

**System.out.println("China: " + chinaFormat.format(payment));**

**System.out.println("France: " + franceFormat.format(payment));**

**}**

**}**

**🧪 Sample Input**

**12324.134**

**✅ Sample Output**

**US: $12,324.13**

**India: ₹12,324.13**

**China: ￥12,324.13**

**France: 12 324,13 €**

**Note: The Indian currency symbol may appear as ₹ or Rs. depending on your system's locale support.**

If the HackerRank test case expects ￥, but the platform doesn't allow it, this might be a **test case issue**. You can raise it with HackerRank support or your test administrator.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

String

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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,  and , perform the following operations:

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

**Input Format**

The first line contains a string . The second line contains another string . 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  and .  
For the second line, write Yes if  is lexicographically greater than  otherwise print No instead.  
For the third line, capitalize the first letter in both  and  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  is "hello" and  is "java".

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

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

import java.io.\*;

import java.util.\*;

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);

    String A=sc.next().toLowerCase();

    String B=sc.next().toLowerCase();

    sc.close();

    System.out.println(A.length()+B.length());

    if(A.compareTo(B)>0){

        System.out.println("Yes");

    }else{

        System.out.println("No");

    }

    System.out.println(A.substring(0,1).toUpperCase()+A.substring(1)+" "+B.substring(0,1).toUpperCase()+B.substring(1));

    }

}

**Lexicographically larger** means comparing strings based on dictionary order — just like how words are sorted alphabetically.

**🔤 Lexicographical Comparison Explained:**

* A string **A** is lexicographically larger than string **B** if it would appear **after** B in a dictionary.
* The comparison is done **character by character** from left to right.
* For example:
  + "banana" is **larger** than "apple" because 'b' > 'a'.
  + "hello" is **not larger** than "java" because 'h' < 'j'.

**✅ In Java:**

You can use:

* If result > 0 → A is lexicographically **larger** than B.
* If result < 0 → A is **smaller**.
* If result == 0 → A and B are **equal**.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

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

**Constraints**

* will consist at most  lower case english letters.

**Sample Input**

madam

**Sample Output**

Yes

import java.io.\*;

import java.util.\*;

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);

     String a=sc.next();

     sc.close();

     StringBuilder sb=new StringBuilder();

    //StringBuilder sb1=new StringBuilder(a);

     sb.append(a);

     sb.reverse();

    // System.out.println("a=: "+a);

     //System.out.println("reverse of a=: "+sb);

     //System.out.println("compare=: "+sb1.equals(sb));

**if(a.equals(sb.toString())){//use tostring of object otherwise it will retuen always false**

        System.out.println("Yes");

     }else{

        System.out.println("No");

     }

    }

}

**Two strings,  and , 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  and  are case-insensitive anagrams, return true. Otherwise, return false.**

**Input Format**

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

**import java.io.\*;**

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

**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);**

**String a=sc.next();**

**String b=sc.next();**

**sc.close();**

**char[]ar=a.toLowerCase().toCharArray();**

**char[] br=b.toLowerCase().toCharArray();**

**Arrays.sort(ar);**

**Arrays.sort(br);**

**if(Arrays.equals(ar, br)){**

**System.out.println("Anagrams");**

**}else{**

**System.out.println("Not Anagrams");**

**}**

**}**

**}**

**A screen shot of a computer

AI-generated content may be incorrect.**

import java.io.\*;

import java.util.\*;

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);

  String line=sc.nextLine();

  sc.close();

**String[] words=line.trim().replaceAll("[^a-zA-Z0-9]+", " ").split("\\s+");**

**if(line.trim().isBlank()){**

**System.out.println("0");**

**}else{**

**System.out.println(words.length);**

**for(String word: words){**

    System.out.println(word);

  }

 }

}

}

Using **Regex**, we can easily match or search for patterns in a text. Before searching for a pattern, we have to specify one using some well-defined syntax.

In this problem, you are given a pattern. You have to check whether the syntax of the given pattern is valid.

**Note**: In this problem, a regex is only valid if you can compile it using the [Pattern.compile](http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html" \l "compile%28java.lang.String%29" \t "_blank) method.

**Input Format**

The first line of input contains an integer , denoting the number of test cases. The next  lines contain a string of any printable characters representing the pattern of a regex.

**Output Format**

For each test case, print Valid if the syntax of the given pattern is correct. Otherwise, print Invalid. Do not print the quotes.

**Sample Input**

3

([A-Z])(.+)

[AZ[a-z](a-z)

batcatpat(nat

**Sample Output**

Valid

Invalid

Invalid

import java.io.\*;

import java.util.\*;

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 linelenght=Integer.parseInt(sc.nextLine());**

         for(int i=0;i<=linelenght-1;i++){

**String word=sc.nextLine();**

**try{**

**Pattern.compile(word);**

**System.out.println("Valid");**

**}catch(Exception ex){**

**System.out.println("Invalid");**

            }

         }

        sc.close();}}

  Write a class called *MyRegex* which will contain a string pattern. You need to write a regular expression and assign it to the pattern such that it can be used to validate an IP address. Use the following definition of an IP address:

IP address is a string in the form "A.B.C.D", where the value of A, B, C, and D may range from 0 to 255. Leading zeros are allowed. The length of A, B, C, or D can't be greater than 3.

Some valid IP address:

000.12.12.034

121.234.12.12

23.45.12.56

Some invalid IP address:

000.12.234.23.23

666.666.23.23

.213.123.23.32

23.45.22.32.

I.Am.not.an.ip

In this problem you will be provided strings containing any combination of ASCII characters. You have to write a regular expression to find the valid IPs.

Just write the MyRegex class which contains a String . The string should contain the correct regular expression.

(MyRegex class *MUST NOT* be public)

**Sample Input**

000.12.12.034

121.234.12.12

23.45.12.56

00.12.123.123123.123

122.23

Hello.IP

**Sample Output**

true

true

true

false

false

false

import java.io.\*;

import java.util.\*;

import java.util.regex.\*;

class MyRegex {

    String pattern =

        "^((25[0-5]|2[0-4][0-9]|1[0-9]{2}|0?[0-9]{1,2})\\.){3}"

      + "(25[0-5]|2[0-4][0-9]|1[0-9]{2}|0?[0-9]{1,2})$";

}

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);

    MyRegex rgx=new MyRegex();

    while(sc.hasNext()){

        String ip=sc.nextLine();

        System.out.println(Pattern.matches(rgx.pattern, ip));

    }

    sc.close();

    }

}

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  to  characters inclusive. If the username consists of less than  or greater than  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 , *uppercase* characters , and digits .
* The *first* character of the username must be an *alphabetic* character, i.e., either *lowercase* character  or *uppercase* character .

For example:

| **Username** | **Validity** |
| --- | --- |
|  | INVALID; Username length < 8 characters |
|  | VALID |
|  | VALID |
|  | INVALID; Username begins with non-alphabetic character |
|  | 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**

* 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.

import java.io.\*;

import java.util.\*;

import java.util.regex.\*;

class UsernameValidator {

    public static final String regularExpression = "^[a-zA-Z][a-zA-Z0-9\_]{7,29}$";

}

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);

Scanner scan = new Scanner(System.in);

        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");

            }

        }

        scan.close();

    }

}

In a tag-based language like *XML* or *HTML*, contents are enclosed between a *start tag* and an *end tag* like <tag>contents</tag>. Note that the corresponding *end tag* starts with a /.

Given a string of text in a tag-based language, parse this text and retrieve the contents enclosed within sequences of well-organized tags meeting the following criterion:

1. The name of the *start* and *end* tags must be same. The HTML code <h1>Hello World</h2> is *not valid*, because the text starts with an h1 tag and ends with a non-matching h2 tag.
2. Tags can be nested, but content between nested tags is considered *not valid*. For example, in <h1><a>contents</a>invalid</h1>, contents is *valid* but invalid is *not valid*.
3. Tags can consist of any printable characters.

**Input Format**

The first line of input contains a single integer,  (the number of lines).  
The  subsequent lines each contain a line of text.

**Constraints**

* Each line contains a maximum of  printable characters.
* The total number of characters in all test cases will not exceed .

**Output Format**

For each line, print the content enclosed within valid tags.  
If a line contains multiple instances of valid content, print out each instance of valid content on a new line; if no valid content is found, print None.

**Sample Input**

4

<h1>Nayeem loves counseling</h1>

<h1><h1>Sanjay has no watch</h1></h1><par>So wait for a while</par>

<Amee>safat codes like a ninja</amee>

<SA premium>Imtiaz has a secret crush</SA premium>

**Sample Output**

Nayeem loves counseling

Sanjay has no watch

So wait for a while

None

Imtiaz has a secret crush

import java.io.\*;

import java.util.\*;

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 scan = new Scanner(System.in);

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

        // Regex to match valid tag content

        String regex = "<([^<>]+)>([^<>]+)</\\1>";

        Pattern pattern = Pattern.compile(regex);

        while (numLines-- > 0) {

            String line = scan.nextLine();

            Matcher matcher = pattern.matcher(line);

            boolean found = false;

            while (matcher.find()) {

                System.out.println(matcher.group(2)); // Print the content between tags

                found = true;

            }

            if (!found) {

                System.out.println("None");

            }

        }

        scan.close();

    }

}