01. Java Map

You are given a phone book that consists of people's names and their phone number. After that you will be given some person's name as query. For each query, print the phone number of that person.

Input Format

The first line will have an integer ***N*** denoting the number of entries in the phone book. Each entry consists of two lines: a name and the corresponding phone number.

After these, there will be some queries. Each query will contain a person's name. Read the queries until end-of-file.

Constraints:

A person's name consists of only lower-case English letters and it may be in the format 'first-name last-name' or in the format 'first-name'. Each phone number has exactly 8 digits without any leading zeros.

$$1 \leq n \leq 100000$$
$$1 \leq Query \leq 100000$$

Output Format

For each case, print "Not found" if the person has no entry in the phone book. Otherwise, print the person's name and phone number. See sample output for the exact format.

To make the problem easier, we provided a portion of the code in the sample code. You can either complete that code or write completely on your own.

Sample Input

```
3
uncle sam
99912222
tom
11122222
harry
12299933
uncle sam
uncle tom
harry
```

Sample Output

```
uncle sam=99912222
Not found
harry=12299933
```

Sample code:

```
//Complete this code or write your own from scratch
import java.util.*;
import java.io.*;

class Solution{

}
```

02. Java Method Overriding (Super Keyword)

When a method in a subclass overrides a method in superclass, it is still possible to call the overridden method using super keyword. If you write super.func() to call the function func(), it will call the method that was defined in the superclass.

You are given a partially completed code in the Sample code. Modify the code so that the code prints the following text:

```
Hello I am a motorcycle, I am a cycle with an engine.
My ancestor is a cycle who is a vehicle with pedals.
```

Sample code:

```
class BiCycle{
      String define_me(){
            return "a vehicle with pedals.";
      }
}

lass MotorCycle extends BiCycle{
      String define_me(){
            return "a cycle with an engine.";
      }

      MotorCycle(){
            System.out.println("Hello I am a motorcycle, I am "+ define_me());
            String temp=define_me(); //Fix this line


            System.out.println("My ancestor is a cycle who is "+ temp );
      }

}
class Solution{
      public static void main(String []args){
            MotorCycle M=new MotorCycle();
      }
}
```

03. Java Abstract Class

A Java abstract class is a class that can't be instantiated. That means you cannot create new instances of an abstract class. It works as a base for subclasses. You should learn about Java Inheritance before attempting this challenge.

Following is an example of abstract class:

```
abstract class Book{
    String title;
    abstract void setTitle(String s);
    String getTitle(){
        return title;
    }
}
```

If you try to create an instance of this class like the following line you will get an error:

```
Book new_novel=new Book();
```

You have to create another class that extends the abstract class. Then you can create an instance of the new class.

Notice that setTitle method is abstract too and has no body. That means you must implement the body of that method in the child class.

In the sample code, we have provided the abstract Book class and a Main class. In the Main class, we created an instance of a class called MyBook. Your task is to write just the MyBook class.

Your class mustn't be public.

Sample Input

```
A tale of two cities
```

Sample Output

```
The title is: A tale of two cities
```

Sample code:

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

04. Java Interface

A Java interface can only contain method signatures and fields. The interface can be used to achieve polymorphism. In this problem, you will practice your knowledge on interfaces.

You are given an interface AdvancedArithmetic which contains a method signature int divisor_sum(int n). You need to write a class called MyCalculator which implements the interface.

divisorSum function just takes an integer as input and return the sum of all its divisors. For example divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12. The value of n will be at most 1000.

Read the partially completed code in the sample code and complete it. You just need to write the MyCalculator class only. Your class shouldn't be public.

Sample Input

```
6
```

Sample Output

```
6
```

Explanation

Divisors of 6 are 1,2,3 and 6. 1+2+3+6=12.

Sample code

```
import java.util.*;
interface AdvancedArithmetic{
  int divisor_sum(int n);
}
//Write your code here


class Solution{

}
```

05. Java Exception Handling (Try-catch)

Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional conditions requiring special processing – often changing the normal flow of program execution. (Wikipedia)

Java has built-in mechanism to handle exceptions. Using the try statement we can test a block of code for errors. The catch block contains the code that says what to do if exception occurs.

This problem will test your knowledge on try-catch block.

You will be given two integers **X** and **Y** as input, you have to compute x/y. If and are not 32 bit signed integers or if **Y** is zero, exception will occur and you have to report it. Read sample Input/Output to know what to report in case of exceptions.

Sample Input 0:

```
10
3
```

Sample Output 0:

```
3
```

Sample Input 1:

```
10
Hello
```

Sample Output 1:

```
java.util.InputMismatchException
```

Sample Input 2:

```
10
0
```

Sample Output 2:

```
java.lang.ArithmeticException: / by zero
```

Sample Input 3:

```
23.323
0
```

Sample Output 3:

```
java.util.InputMismatchException
```

06. Simple Array Sum

Given an array of integers, find the sum of its elements.

Input Format

The first line contains an integer, **N** , denoting the size of the array.

The second line contains **N** space-separated integers representing the array's elements.

Output Format

Print the sum of the array's elements as a single integer.

Sample Input

```
6
1 2 3 4 10 11
```

Sample Output

```
31
```

Explanation

We print the sum of the array's elements: $1 + 2 + 3 + 4 + 10 + 11 = 31$.

07. Two Characters

In this challenge, you will be given a string. You must remove characters until the string is made up of any two alternating characters. When you choose a character to remove, all instances of that character must be removed. Your goal is to create the longest string possible that contains just two alternating letters.

As an example, consider the string abaacdabd. If you delete the character a, you will be left with the string bcdbd. Now, removing the character c leaves you with a valid string bdbd having a length of 4. Removing either b or d at any point would not result in a valid string.

Given a string **S**, convert it to the longest possible string **T**, made up only of altrenating characters. Print the length of string **S** on a new line. If no string **T** an be formed, print **0** instead.

Input Format

The first line contains a single integer denoting the length of **S**.

The second line contains string **S**.

Constraints

- $1 \leq |s| \leq 1000$
- $s[i] \in \text{ascii[a-z]}$

Output Format

Print a single integer denoting the maximum length of T for the given S if it is not possible to form string t, print 0 instead.

Sample Input

```
10
beabeefeab
```

Sample Output

```
5
```

Explanation

The characters present in S are a, b, e, and f. This means that T must consist of two of those characters and we must delete two others. Our choices for characters to leave are [a,b], [a,e], [a, f], [b, e], [b, f] and [e, f].

If we delete e and f, the resulting string is babab. This is a valid T as there are only two distinct characters (a and b), and they are alternating within the string.

If we delete a and f, the resulting string is bebeeeb. This is not a valid string T because there are consecutive e's present. Removing them would leave consecutive b's, so this fails to produce a valid string T

Other cases are solved similarly.

babab is the longest string we can create..

08. Java Singleton Pattern

"The singleton pattern is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system

Complete the Singleton class in your editor which contains the following components:

1. A private Singleton non parameterized constructor.

2. A public String instance variable named ***str***.

3. Write a static method named getSingleInstance that returns the single instance of the Singleton class.

09. Java Factory Pattern

According to Wikipedia, a factory is simply an object that returns another object from some other method call, which is assumed to be "new".

In this problem, you are given an interface *Food* . There are two classes *Pizza* and *Cake* which implement the *Food* interface, and they both contain a method *getType ()*.

The main function in the *Main* class creates an instance of the *FoodFactory* class. The *FoodFactory* class contains a method *getFood(String)* that returns a new instance of *Pizza* or *Cake* according to its parameter.

You are given the partially completed code in the editor. Please complete the *FoodFactory* class.

Sample Input 1

```
cake
```

Sample Output 1

```
The factory returned class Cake
Someone ordered a Dessert!
```

Sample Input 2

```
pizza
```

Sample Output 2

```
The factory returned class Pizza
Someone ordered Fast Food!
```

10. Java List

For this problem, we have 2 types of queries you can perform on a List:

1. Insert **Y** at index **X**:
2. Delete the element at index **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.

Constraints:

- $1 \leq N \leq 4000$
- $1 \leq Q \leq 4000$
- Each element in is a 32-bit integer

Output Format

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

11. Java Generics

Generic methods are a very efficient way to handle multiple datatypes using a single method. This problem will test your knowledge on Java Generic methods.

Let's say you have an integer array and a string array. You have to write a **single** method *printArray* that can print all the elements of both arrays. The method should be able to accept both integer arrays or string arrays.

Do not use method overloading because your answer will not be accepted.

12. String Reverse

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

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

Constraints

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

Sample Input 1

```
madam
```

Sample Output 1

```
Yes
```

13. Insert a node at a specific position in a linked list

You're given the pointer to the head node of a linked list, an integer to add to the list and the position at which the integer must be inserted. Create a new node with the given integer, insert this node at the desired position and return the head node. A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The head pointer given may be null meaning that the initial list is empty.

Input Format

You have to complete the **Node* Insert(Node* head, int data, int position)** method which takes three arguments - the head of the linked list, the integer to insert and the position at which the integer must be inserted. You should NOT read any input from stdin/console. **position** will always be between 0 and the number of the elements in the list (inclusive).

Output Format

Insert the new node at the desired position and return the head of the updated linked list. Do NOT print anything to stdout/console.

Sample Input

NULL, data = 3, position = 0

3 --> NULL, data = 4, position = 0

Sample Output

```
3 --> NULL
4 --> 3 --> NULL
```

Explanation

1. we have an empty list and position 0. 3 becomes head.

2. 4 is added to position 0, hence 4 becomes head.