CS61B, Fall 2015 **Lab #i8** **P. N. Hilfinger**

Due: Fri., 30 October 2015

Rather than have a lab devoted to current topics in the course, we're going to have a problem-solving lab instead. You won't be graded on how much you complete successfully (as usual), but we do suggest giving it a try. The problems here come from various programming contests. The idea in these contests is to work for speed of programming. Some teams of three at the ACM International Programming Contest, for example, will solve 12 (or even 13) problems like these in five hours.

In all of these problems, the input will come from the standard input and output will go to the standard output.

You'll find (trivial) skeletons and some test data for your answers in the `lab8` staff directory.

1. [From the Valladolid archives] An imaging device furnishes digital images of two machined surfaces that eventually will be assembled in contact with each other. The roughness of this final contact is to be estimated.

A digital image is composed of the two characters, '`X`' (marking places where material is present) and '`␣`' (blank, indicating space). The first column will always have an '`X`' in it and will be part of the left surface. The left surface can extend to the right as contiguous `X`s. Similarly, column N will always have an '`X`' in it and will be part of the right surface, where N is furthest right position of any '`X`' over all rows. The right surface can extend to the left from column N as contiguous `X`s. Both surfaces will have the same vertical extent (the same number of rows), so that there is a section of left surface and right surface on each row of the image.

For example, here is a possible image (with blanks shown as '`␣`':

```
XXXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXXX

XXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXXXXX

XXXXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXX
```

Left Surface **Right Surface**

```
XX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXXXX

XXXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXX

XXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXXXX
```

In this example, each line is 30 characters long

In each row of the image, there can be zero or more blanks separating the left surface from the right surface. There will never be more than a single blank region in any row.

For each image given, you are to determine the total empty area—the total number of blanks between the surfaces—that will exist after the left surface has been brought into contact with the right surface. The two surfaces are brought into contact by displacing them strictly horizontally towards each other until a rightmost '`X`' of the left surface of some row is immediately to the left of the leftmost '`X`' of the right surface of that row. There is no rotation or twisting of these two surfaces as they are brought into contact; they remain rigid, and only move horizontally.

The input consists of a series of digital images. Each image consists of one or more lines of `X`s and blanks, all having the same length and all beginning and ending with `X`. There will be an empty line after every data set.

For each image in the series, output the total empty area, using the format shown in the example.

Example:

Input	Output
`XXXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXXX`	`Image 1: 4`
`XXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXXXXX`	
`XXXXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXX`	`Image 2: 0`
`XX␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣XXXXXX`	
	`Image 3: 0`
`XXXXXXXXXXXXXXXXXXXXXXXXX`	
`XXXXXXXXXXXXXXXXXXXXXXXXX`	
`XXXXXXXXX␣␣␣␣␣␣␣␣␣␣␣␣␣␣XX`	

2. [Adapted from the Valladolid archives] Consider binary trees whose nodes are labeled with single digits and letters (upper- and lowercase, case-sensitive). Given the preorder (root, left subtree, right subtree) node order and the inorder (left subtree, root, right subtree) node order for the same tree, it is possible to reconstruct the tree, assuming no two nodes in the tree have the same label. For example, given the preorder traversal order “DBACEGF” and the inorder traversal order “ABCDEFG,” you can compute that the tree these come from is

```
      D
     / \
    /   \
   B     E
  / \     \
 /   \     \
A     C     G
           /
          /
         F
```

You are to write a program to do this reconstruction on any such tree.

The input consists of one or more cases in free format. Each case consists of two non-empty strings *PRE* and *IN*, representing the preorder traversal and inorder traversal of a non-empty binary tree, and consisting of letters and digits. Case is significant.

For each test case, print the reconstructed tree’s postorder traversal (left subtree, right subtree, root), using the format shown in the example.

Example:

Input	Output
`DBACEGF`	`Case 1: ACBFGED`
`ABCDEFG`	
`BCAD CBAD`	`Case 2: CDAB`

3. [Due to E. W. Dijkstra] Consider decimal numerals containing only the digits 1–3. A numeral is considered "good" if no two adjacent non-empty substrings of it are equal; otherwise it is "bad." Hence, the numerals '1', '12', and '1213' are good, while '11', '32121', and '121321312' are bad.

You are to write a program that, given $n > 0$, finds the smallest good n-digit numeral. The input consists of a sequence of positive integers. For each of these integers, n, the output is to contain a line of the form

`The smallest good numeral of length` n `is` s`.`

where s is the answer. For example,

Input	Output
`1 4`	`The smallest good numeral of length 1 is 1.`
`7`	`The smallest good numeral of length 4 is 1213.`
`9`	`The smallest good numeral of length 7 is 1213121.`
	`The smallest good numeral of length 9 is 121312313.`