

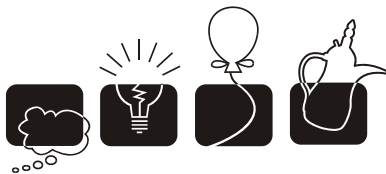
The 28th Annual ACM International Collegiate Programming Contest

Arab and North Africa 6th Regional Contest

sponsored by:
Kuwait University
Kuwait Foundation for the Advancement of Science
IBM, NCS, GBM

Kuwait University
December 4, 2003

- The problem set is made of 16 numbered pages
- The maximum allowable running time for a submission is 60 seconds.



[A] Isomorphic Numbers

Program:	isonum.(c cpp java)
Input:	isonum.in

Description

Two positive decimal integers without any leading zeros will be called isomorphic if they are composed of the same number of digits and same digit-repetition pattern (regardless of the value of the digits.)

For example, all of the following numbers are isomorphic to each other: 12156, 73748, 51590, 48412 since in each individual number, the first and the third positions are occupied by the same digit, and all the other positions have distinct digits (which are also distinct from the digit in positions 1 and 3).

The following numbers are also isomorphic to each other: 237392, 578715, 341453. All of the following numbers are pair-wise not isomorphic: 222, 545, 776, 811, 66.

The set of numbers that are isomorphic to each other will be called an *isoret*. For example, the numbers: 11, 22, 33, 44, 55, 66, 77, 88, 99 form an isoret with cardinality 9 (number of elements in the set). Any number is a member of exactly one isoret.

A company wants to make use of isomorphic numbers in classifying its clients into groups. Each client is given a unique id number (a decimal positive number less than 1,000,000,000.) Any two clients within the same group are given id numbers that are isomorphic to each other. The company hires you to write a program that will do two things:

1. Given two numbers, determine if the numbers are isomorphic or not.
2. Given a number x , determine the cardinality of its isoret.

Input Format

The program accepts a list of one or more *commands*. Each command is specified on a separate line. Possible commands are:

VERIFY x y	Test if numbers x and y are isomorphic or not.
SIZE x	Print the cardinality of the isoret x is a member of.
QUIT	Exit the program.

Commands are always uppercase words.

Output Format

For a VERIFY x y command, your program should output a line of the form:

k . \square result

where result is 'true' (without the quotes,) if x and y are isomorphic, 'false' otherwise. k is the input line number of the command. \square is a single space character.

For the SIZE x command, your program should print:

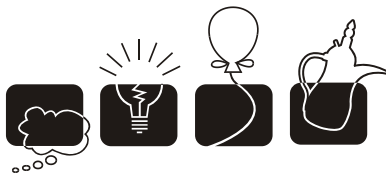
k . \square c

Where c is the cardinality.

Sample Input/Output

```
isonum.in  
VERIFY 1121321 5545645  
VERIFY 1111 1112  
SIZE 11  
QUIT
```

```
OUTPUT  
1. true  
2. false  
3. 9
```



[B] Microsoft Word

Program:	spell.(c cpp java)
Input:	spell.in

Description

Microsoft is a startup software company with a killer word processor. All it needs is a spell checker to take over the word-processing market. They believe they have the concept for a perfect spell checker but they can't program it, and this is where you come in.

Microsoft wants you to write a spell checker that takes a dictionary of valid words, and a list of words to spell check. Let w be a word being spelled checked; let n be the length of w . Let D be the set of words in the dictionary and d be a word in the dictionary.

1. If $w \in D$, then your program should print w as is. Otherwise,
2. if there is exactly one $d \in D$ such that w is missing exactly one letter from d , then the spell checker should print d . However,
3. If there are more than one word $d \in D$, that meet criteria 2 above, then the spell checker should print a string made of n question-marks ('?').
4. Finally, if all the above fails, the spell checker should print a string made of n stars ('*').

For example, given the dictionary $D = \{\text{rice}, \text{price}, \text{prize}\}$, the following table illustrates what your program should print for the words **rice**, **ice**, **prie**, and **ize**.

input	output	justification
rice	rice	case 1: $w \in D$
ice	rice	case 2: exactly one word is found by adding exactly one character to w .
prie	????	case 3: there are more than one word (price , prize) that can be formed by adding one character to w .
ize	***	case 4.

Input Format

Your program will be run against a number of test cases. The first line in the input file contains a single integer T specifying the number of test cases.

Each test case is made of two lists of words: The list of words in the dictionary is specified first, followed by the list of words to spell check. Each word is specified on a separate line. The end of the dictionary is indicated using the string '---' on a separate line. The end of the words to spell check is indicated using the string '+++'. All words are of lower-case letters, and no word is longer than 32 characters. The dictionary will have at most 100,000 words.

Output Format

At the beginning of each test case, your program should print the following line:

```
Test Case #k
```

where *k* is the test case number. Test cases are numbered starting at 1. Following the first line, for each word in the list of words to be spell checked, write on a separate line the output of your spell checker using the following format:

```
 w=>r
```

Where *w* is the word being spelled checked, and *r* is the output word from the spell checker. Note the spaces before *w* and around the arrow '*=>*'.

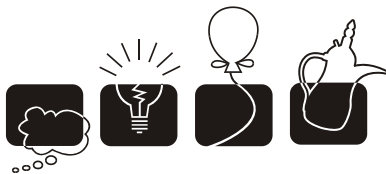
A blank line should be added after the end of each test case.

Sample Input/Output

```
spell.in
2
rice
price
prize
---
rice
ice
prie
ize
+++
seen
been
teen
---
see
een
ten
bye
+++
```

```
OUTPUT
Test Case #1
rice => rice
ice => rice
prie => ????
ize => ***

Test Case #2
see => seen
een => ???
ten => teen
bye => ***
```



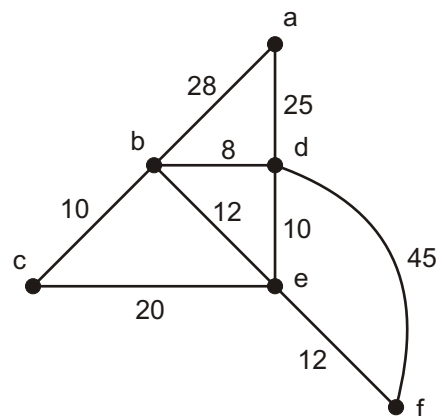
[C] Johnny the Taxi Driver

Program:	taxi.(c cpp java)
Input:	taxi.in

Description

Johnny is a dishonest taxi driver who just loves to get more money out of unaware tourists. When he picks up a tourist, Johnny always attempts to drive more than necessary in order to be able to charge the tourist more. However, Johnny realizes that even tourists visiting the city for the first time, will be able to uncover his scheme if he keeps going around the same places too many times. Johnny must find paths with minimum revisits of the same places, and never more than 10 revisits. For that Johnny needs your help.

Johnny will give you a detailed description of the city, listing distances between intersection points. Johnny wants you to write a program that will determine a path between any two given points, such that the path will have a preset cost. For example: In the map on the right, Rather than going from a to f using the cheapest (shortest) path adef (with cost= 47) Johnny, being greedy, can go abcebdef or adebdf both with cost= 100. The second path adebdf revisits only one point (d) is better than the first abcebdef which has two revisits (b and e.)



Input Format

Your program will be tested on a number of test cases. The first line of the input file contains a single integer T denoting the number of test cases.

Each test case is specified on $n + q + 1$ lines. The first line of each test case specifies three integers: p (the number of intersection points,) n (the number of streets,) and q (the number of queried paths.)

Following that, the streets will be specified using n lines, one for each street. A street between PointA and PointB with cost C is specified using the following format:

PointA_PointB_C

Each test case has at least one street and at most 100 streets. All streets are two ways. A Point is specified using a lower-case word whose length doesn't exceed 8 characters. No city will have more than 100 distinct points. C is a positive integer ≤ 100

Following the n input lines describing the streets, each test case will have q queries, each specified on a separate line using the following format:

PointX_PointY_Q

which inquires about the minimum number of revisits in a path from PointX to PointY having cost Q.

Output Format

For each query, write on a separate line, the following output:

PointX PointY Q R

Where PointX, PointY, and Q are the same as the input, and R is the minimum number of revisits in a path going from PointX to PointY with cost Q. If there is no such path with no more than 10 revisits, Then R is equal to -1.

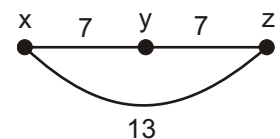
Sample Input/Output

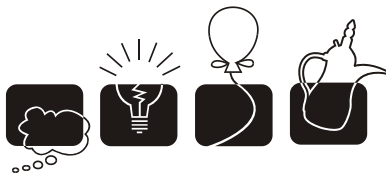
```
taxi.in
2
6 9 2
a b 28
a d 25
b c 10
b d 8
b e 12
c e 20
d e 10
d f 45
e f 12
a f 100
a f 50
3 3 1
x y 7
z y 7
z x 13
x z 65
```

```
OUTPUT
a f 100 1
a f 50 -1
x z 65 4
```

Note:

The second test case uses the graph on the right and inquires about a path from x to z with cost=65. This can only be achieved using the path xzxxzx which has 4 revisits (x is revisited twice, and so is z).





[D] In Memory of Denis Roskin

Program:	pair.(c cpp java)
Input:	pair.in

Description

Denis Roskin, Jr. has come up with an interesting mathematical series in memory of his late father, Dr. Denis Roskin, Sr., the long forgotten mathematician who died at the age of 40. Actually, Jr. came up with two (rather coupled) series, which he named *The Denis Roskin Pair* or DR for short. Each is defined as follows:

The Denis Series

$$D(0) = 1$$

$$D(i) = \sum_{k=1}^i \left\lfloor \frac{R(i-k)}{k} \right\rfloor \quad \text{where } i > 0$$

The Roskin Series

$$R(0) = 1$$

$$R(i) = \sum_{k=0}^{i-1} \left\lceil \frac{D(k)}{i-k} \right\rceil \quad \text{where } i > 0.$$

where $\lfloor x \rfloor$ is the floor of x and $\lceil x \rceil$ is the ceiling of x . The first 10 elements in the series are:

index:	i	0	1	2	3	4	5	6	7	8	9
Denis Series:	$D(i)$	1	1	1	2	4	6	11	17	29	47
Roskin Series:	$R(i)$	1	1	2	3	5	8	12	20	30	49

Write a program to compute the Denis Series.

Input Format

Your program will be tested on a number of test cases. The input is made of one or more integers v_i terminated by a -1 (which is not part of the test cases.) Each integer appears on a separate line. Note that $0 \leq v_i \leq 40$ (remember: Dr. Denis Roskin died at the age of 40).

Output Format

For each v_i in the input, write on a separate line, the value of $D(v_i)$.

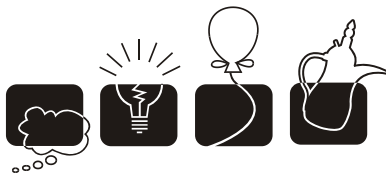
Sample Input/Output

pair.in

```
1
5
9
-1
```

OUTPUT

```
1
6
47
```



[E] The Game of Nubble

Program:	nubble.(c cpp java)
Input:	nubble.in

Description

The game of Nubble is a very popular game among math teachers in British elementary schools. It is used as a fun way to teach basic arithmetic to kids. In this game, a player throws a set of two or more dice (normally four, never more than six,) and tries to come up with an arithmetic expression using *all the values* on the dice they got. Expressions are formed using the basic four arithmetic operations: addition, subtraction, multiplication, and *even* division¹. The challenge is to get the largest possible value less than 100.

For example, imagine getting the four dice values: 2, 6, 3, and 1. One can make the expression $(6 + 3 + 2 + 1) = 12$. But we can do better than that since $(6 * 3 * 2 + 1) = 37$. But the best value you can get given these dice is $54 = 6 * 3 * (1 + 2)$.

Write a program that takes a list of dice values and determines an arithmetic expression with the largest value less than 100.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line using the form:

$$n \ v_1 \ v_2 \ \dots \ v_n$$

Where n is the number of dice, and $v_1 \dots v_n$ are the dice values. The end of test cases is indicated by the value zero on a separate line.

Output Format

For each test case, print the best value that can be achieved on a separate line.

Sample Input/Output

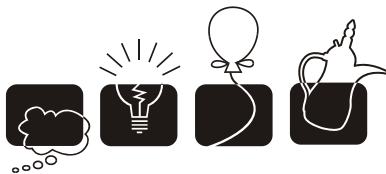
nubble.in

4 2 6 3 1
3 5 5 6
0

OUTPUT

54
60

¹Division is allowed only if the dividend is completely divisible by the divisor, i.e. the remainder is zero.



[F] Traffic Control

Program:	police.(c cpp java)
Input:	police.in

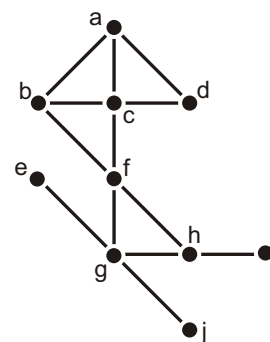
Description

Kuwait's Traffic Police is considering acquiring new, state of the art, speed cameras (cameras that take pictures of cars driving over the speed limit). Unlike the first generation which had to be pointed to a fixed direction, these cameras can rotate and take pictures in any direction. Placed at a road intersection, one new camera can monitor all roads passing through that intersection. In the figure below, a camera placed at intersection *f* covers the streets *acfg* and *bfh*

But these cameras are quite expensive and the police department needs to determine the minimum number of cameras it must buy.

For example, in a city with a road map similar to the one on the right, one can place four cameras at road intersections *a*, *c*, *f* and *g*, to cover all the streets. The police department can save money, however, by buying only three of these cameras and placing them at intersections *b*, *d*, and *g* (or *a*, *b*, and *g*.)

Write a program that takes a map of the roads in a city and computes the minimum number of cameras needed to monitor all the roads.



Input Format

Your program will be tested on a number of test cases. The first line of the input will have a single integer *T* representing the number of test cases. Each test case is specified on a single line using the form:

$$n \text{ street}_1 \text{ street}_2 \dots \text{street}_n$$

Where *n* is the number of streets with $0 \leq n \leq 50$. Each street is specified using a string representing the intersections that street passes through. Each intersection is represented as a single, lowercase letter ('a' through 'z', inclusive.)

Output Format

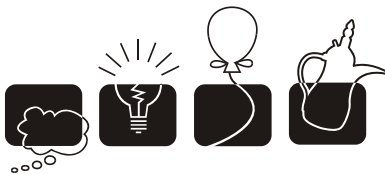
For each test case, print the minimum number of cameras needed to monitor all the streets in the given test case.

Sample Input

_____ police.in _____
1
7 ab acfg ad bcd bfh egj ghi

Sample Output

_____ OUTPUT _____
3



[G] Who Is the Real Second

Program:	second.(c cpp java)
Input:	second.in

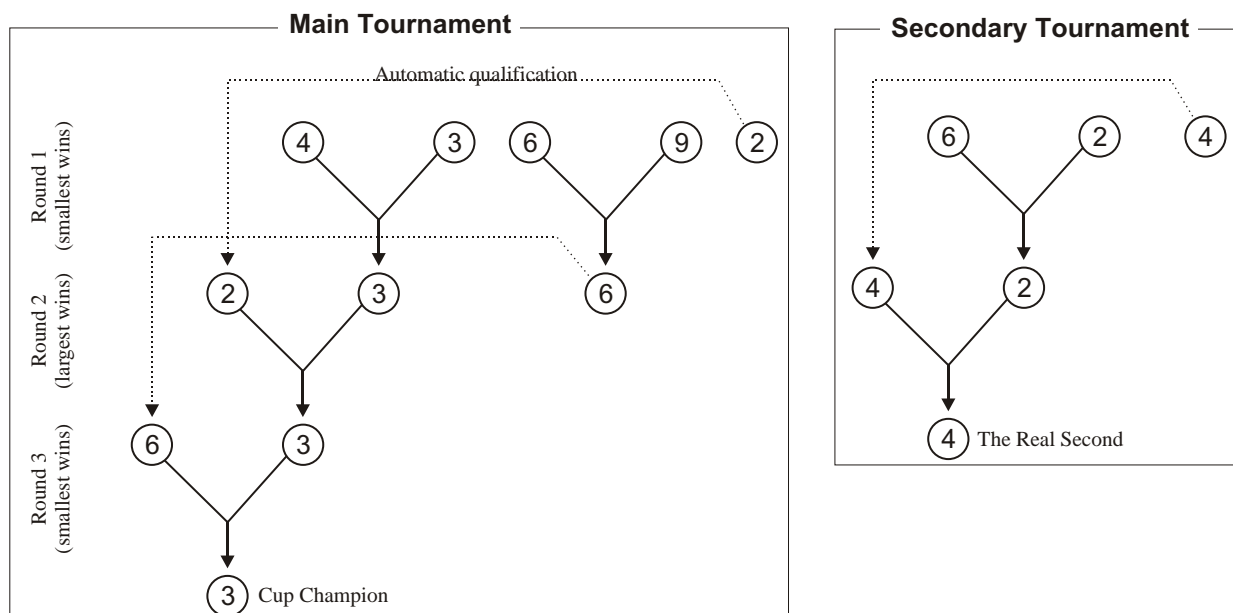
Description

There are two methods to determine winners of football tournaments: league or cup. In the cup method, football matches are played in *rounds*. In the first round, every two teams play one match and only the winner qualifies to the next round. The winners of a round are then reduced by half using the same procedure. This continues until the final round where two teams play and the winner is declared the cup champion. The loser of this final match is normally considered the second. Some argue that this is unfair and that the real second team should be the best team of *all the teams that lost against the champion*.

Write a program to determine the (real) second team in the following tournament: There are n teams denoted by n distinct integer identifiers. These n teams will play in a cup manner until the champion is determined. The winner in any match is the team with the smaller identifier (if the round number is odd,) or the team with the larger identifier (if the round number is even.) If in some round, the number of teams is odd, the last team in the list is qualified automatically to the next round, but is placed in the head of the next list (in the example below, team 2 qualifies automatically to round 2. Similarly, team 6 to round 3.)

Once the champion is determined, there will be another tournament (the secondary tournament) for all the teams *that lost against the champion*. These teams will be placed in the list in reverse order so that the team that lost to the champion in the final round is placed first, and so on. These teams will play again in a cup manner with the same rules used before. The winner this time is the real second.

Take a look at this tournament in which five teams (ids: 4, 3, 6, 9, and 2) compete, and team 3 is the cup champion. Teams that lost to team 3 (6, 2, and 4) play in the secondary tournament to determine the team in the second place. The real second is team 4, not 6.



Input Format

Your program will be tested on multiple test cases. Each test case is specified on a single text line using the form:

$$n \text{ id}_1 \text{ id}_2 \dots \text{id}_n$$

where n is the number of teams, and id_i is the identifier for the i th team. Note that $1 < n < 10,000$ and $-10,000 < \text{id}_i < 10,000$.

The end of the test cases is indicated with a 0 on a separate line.

Output Format

For each test case, write on a separate line, the identifier of the real second.

Sample Input

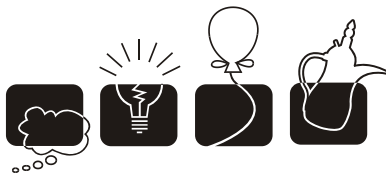
second.in

```
5 4 3 6 9 2
6 10 80 30 20 15 25
0
```

Sample Output

OUTPUT

```
4
20
```



[H] The Game of Yazoo

Program:	yazoo.(c cpp java)
Input:	yazoo.in

Description

Yazoo is a computer game in which you're given a sequence of integers. A sequence can be broken into one or more subsequences (or subranges) such that the numbers of each subsequence satisfy some arithmetic property. Each subsequence is then given a certain score depending on the property it satisfies and on its length. The properties and their scores are:

Sorted: A subsequence of length n in which all its numbers are sorted (all non-decreasing or all non-increasing,) is given the score $5 * n$.

Odd/Even: A subsequence of length n in which its numbers are all even (or all odd,) is given the score $10 * n$.

Alternating: A subsequence of length n in which its numbers alternate between even and odd, is given the score $15 * n$.

Quads: A subsequence of length n in which all its numbers are multiple of 4, is given the score $20 * n$.

Note the following:

1. A subsequence must be contiguous (similar to substrings of strings). For example, in the sequence [1, 2, 3, 4], [1, 2, 3] is a subsequence whereas [1, 2, 4] isn't.
2. The minimum length of a subsequence is 3.
3. Any number in the original sequence can be a member of at most one subsequence.
4. It is possible to have some numbers of the original sequence that are not members of any subsequence. In this case, these numbers don't add anything to the score.

The objective of the game is to break the sequence into subsequences with the maximum total score. For example, the sequence [12, 14, 21, 26, 13, 11, 8] can be broken into the two subsequences [12, 14, 21, 26] (sorted) and [13, 11, 8] (sorted) with a total score of $5 * 4 + 5 * 3 = 35$. However, it is better to choose the subsequence [14, 21, 26, 13] (alternating) as its score is $15 * 4 = 60$.

Write a program that determines the maximum total score a sequence can have.

Input Format

Your program will be tested on a number of test cases. The first line of the input file contains an integer T representing the number of test cases in the input file.

Each test case is described on a single line as a list of integers. The first integer n , is the length of the sequence, and will be followed by n integers, all separated by one or more spaces. A sequence may have up to 1,000 numbers and each number v is in the range $-10,000 < v < +10,000$.

Output Format

For each sequence, write on a separate line, the maximum score you can get for that sequence.

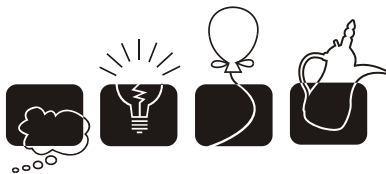
Sample Input/Output

yazoo.in

```
2
7 12 14 21 26 13 11 8
8 6 5 4 8 12 13 14 15
```

OUTPUT

```
60
105
```



[I] XML+XPath=XSLT

Program:	xpath. (c cpp java)
Input:	xpath.in

Description

XML, the Extensible Markup Language, can be used to represent tree-structured data as text documents. Each tree can be described using an XML *element*. The element for a tree with root node R and subtrees S_1, S_2, \dots, S_n as its children, can be represented in XML using the string:

$$\langle R \rangle E_1 E_2 \dots E_n \langle /R \rangle$$

where E_i is the XML element for subtree S_i . Notice that each terminal node (leaf node) has a data item and property describing the data. For example, The tree on the right has a terminal node indicating that the NAME of the PERSON is `linus`. In general, trees can be described using the following grammar: (remember that `tree+` means a sequence of one or more `trees`.)

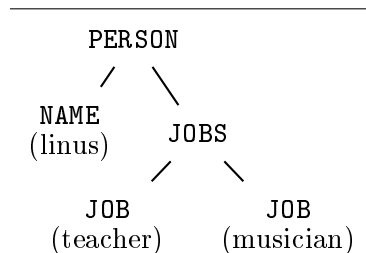
```
tree  → <TAG> data </TAG>
tree  → <TAG> tree+ </TAG>
```

where `data` is a single word, `TAG` is a single word, and the same word used for the opening tag `<TAG>` is used for the closing tag `</TAG>`

An XML document can be viewed as a *forest* with zero or more *trees*. XPATH is a language used to navigate in the trees of an XML document. An `xpath` is made of one or more *locators* separated by slashes. A locator can be a tag name, a positive integer, or the `'*'` character. For example, `/PERSON/*/1` is an `xpath` made of three locators. A tag-name locator selects subtrees having the specified tag. A positive integer k selects the k th subtree, while a `*` selects all subtrees.

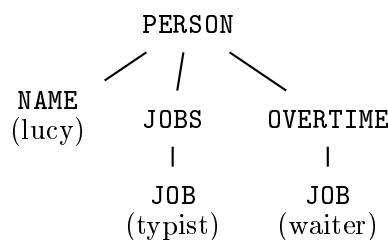
The first locator of an `xpath` is used to select a set of trees from the forest (possibly empty). Each additional locator is used to select subtrees from the (sub)trees already selected by the previous locator(s). For example, `/PERSON` selects all trees with root node `PERSON`, while `/PERSON/NAME` selects the all the `NAME` subtrees of all `PERSON` trees, and so on.

Once the set of subtrees identified by an `xpath` is identified, your program should print the data associated with *all terminal nodes* in that set. Non-terminal nodes are not printed. For example, the `xpath` `/PERSON/1` prints `linus` and `lucy` whereas `/PERSON/*/2` prints only `musician`. Study the sample input/output for more examples.



```

<PERSON>
<NAME>linus</NAME>
<JOBS>
  <JOB>teacher</JOB>
  <JOB>musician</JOB>
</JOBS>
</PERSON>
    
```



```

<PERSON>
<NAME>lucy</NAME>
<JOBS>
  <JOB>typist</JOB>
</JOBS>
<OVERTIME>
  <JOB>waiter</JOB>
</OVERTIME>
</PERSON>
    
```

Input Format

Your program will be tested on one or more test cases. Each test case is made of two sections: The XML document and the list of `xpaths` to run on that document. The beginning of a document is indicated with the string "`*DOCUMENT`" appearing at the beginning of a line by itself. Similarly, the beginning of the `xpaths` section is identified by "`*XPATH`". The end of a test case is identified by the appearance of a "`*DOCUMENT`" keyword (signaling the beginning of the next test case). Or, the appearance of the string "`*QUIT`" (signaling the end of all test cases).

The "`*DOCUMENT`" section is made of zero or more trees. The document section is a free form of which there can be extra spaces and lines (see the sample I/O). Tags are case insensitive, while data is case sensitive.

The "`*XPATH`" section is made of one or more `xpaths`. Each `xpath` is specified completely on a single line. No spaces are allowed within an `xpath`. There might be empty lines within the `XPATH` section (which should be ignored).

Output Format

For each document, your program should print the following line:

```
PROCESSING_DOCUMENT_#_k
```

Where `k` is the document number (starting at 1).

Following that, for each `xpath`, your program should print the following fragment:

```
xpath  
==>_data1  
==>_data2  
...  
==>_dataN
```

Where `xpath` is the `xpath` being processed (using all uppercase letters). `data1...dataN` is the set of data within the current document matching the `xpath` being processed. Notice the empty line *after* the end of output lines generated by `xpath`.

Sample Input

xpath.in

```
*DOCUMENT
<person>
  <NAME>Linus</NAME>
  <JOBS><JOB>teaCHer</JOB><JOB>MUSICian</JOB></JOBS>
</PErSON>

<PERSON>
  <name>lucy</NAME><JOBS><JOB>typist</job></JOBS>
  <OVERTIME><JOB>WAITer</JOB></OVERTIME>
</PERSON>
*XPath

/person/1
/PERson/*/2

*DOCUMENT
<A><B><C>Apple</C><D>Orange</D></B><B>Grape</B><C><D>Banana</D></C></A>
<C><B>Melon</B><A>Kiwi</A></C>
*XPath
/A/B
/*/B
/A/*/D
/1/1/2
/1/2/3/4
*QUIT
```

Sample Output

OUTPUT

```
PROCESSING DOCUMENT # 1
/PERSON/1
==> Linus
==> lucy

/PERSON/*/2
==> MUSICian

PROCESSING DOCUMENT # 2
/A/B
==> Grape

/*/B
==> Grape
==> Melon

/A/*/D
==> Orange
==> Banana

/1/1/2
==> Orange

/1/2/3/4
```