

2006 WSU ACM Programming Contest

Rules

1. All questions require you to read the test data from standard input and write results standard output; You cannot use files for input or output.
2. All programs will be re-compiled prior to testing with the judges' data.
3. Non-standard libraries cannot be used in your solutions. The Standard Template Library and C++ string libraries are allowed.
4. Programming style is not considered in this contest. You are free to code in whatever prefer. Documentation is not required.
5. All communication with the judges will be handled by the PC^2 environment.
6. The allowed programming languages are C, C++ and Java.
7. Judges decisions are to be considered final. No cheating will be tolerated.
8. There are 10 questions to be completed in the allotted time.

1 Climbing Worm

An inch worm is at the bottom of a well n inches deep. It has enough energy to climb u inches every minute, but then has to rest a minute before climbing again. During the rest, it slips down d inches. The process of climbing and resting then repeats. How long before the worm climbs out of the well? Well always count a portion of a minute as a whole minute and if the worm just reaches the top of the well at the end of its climbing, we'll assume the worm makes it out.

Input

There will be multiple problem instances. Each line will contain 3 positive integers n , u and d . These give the values mentioned in the paragraph above. Furthermore, you may assume $d < u$ and $n < 100$. A value of $n = 0$ indicates end of output.

Output

Each input instance should generate a single integer on a line, indicating the number of minutes it takes for the worm to climb out of the well.

Sample Input

```
10 2 1
20 3 1
0 0 0
```

Sample Output

```
17
19
```

2 Ride My Bicycle

Alan Chester Mason, worried about the overuse of fossil fuels and his own expanding waist, has decided to purchase a bicycle. The model he has decided on has 3 chain rings on the crank (the gears attached to the pedal mechanism) and 7 gears on the back wheel. There is a chain connecting one front chain ring to one back gear. This selection can be changed (via the derailleurs) to any combination of front chain ring and back gear desired. The ease of pedalling is determined by the ratio of teeth in the chosen chain ring and the chosen back ring. The smaller the ratio, the easier it is to pedal. Alan has many options for the number of teeth in each chain ring and each back gear and wishes to see what the gearing (that is, ratios) would be for various combinations. You are to write a program to help him out.

Input

Input for this program will be the description of one configuration of chain rings and back gears. Input will be specified on two lines. The first line will have three positive integers, separated by single spaces, in strictly increasing order giving the number of teeth on each chain ring. The second line will contain 7 positive integers, separated by spaces, in strictly increasing order giving the number of teeth on each back gear. We will refer to the chain rings as chain rings 1, 2 and 3 from smallest to largest. We refer to the back gears in a similar manner.

Output

You should generate 21 lines of output for this program. Each line should give information about one particular combination of chain ring and back ring and should be of the form

x.xx f b

where f is the number of the chain ring, b is the number of the back ring and $x.xx$ is the ratio of (teeth in f)/(teeth in b) given to 2 decimal places. These numbers are separated by a single space. These lines are to be printed from smallest ratio to largest. If two (or more) ratios are equal, print the one with the smallest chain ring first. You may assume that the number of teeth are such that the ratios will all be less than 10.

Sample Input

```
32 42 52
12 15 17 18 21 24 28
```

Sample Output

1.14 1 7
1.33 1 6
1.50 2 7
1.52 1 5
1.75 2 6
1.78 1 4
1.86 3 7
1.88 1 3
2.00 2 5
2.13 1 2
2.17 3 6
2.33 2 4
2.47 2 3
2.48 3 5
2.67 1 1
2.80 2 2
2.89 3 4
3.06 3 3
3.47 3 2
3.50 2 1
4.33 3 1

3 TEX Quotes

TEX is a typesetting language developed by Donald Knuth. It takes source text together with a few typesetting instructions and produces, one hopes, a beautiful document. Beautiful documents use double-left-quote and double-right-quote to delimit quotations, rather than the mundane " which is what is provided by most keyboards. Keyboards typically do not have an oriented double-quote, but they do have a left-single-quote ‘ and a right-single-quote ’. Check your keyboard now to locate the left-single-quote key ‘ (sometimes called the “backquote key”) and the right-single-quote key ’ (sometimes called the “apostrophe” or just “quote”). Be careful not to confuse the left-single-quote ‘ with the “backslash” key \ . TEX lets the user type two left-single-quotes ‘ ‘ to create a left-double-quote and two right-single-quotes ’ ’ to create a right-double-quote. Most typists, however, are accustomed to delimiting their quotations with the un-oriented double-quote ". If the source contained

```
"To be or not to be," quoth the bard, "that is the question."
```

then the typeset document produced by TEX would not contain the desired form: ‘ ‘To be or not to be,’ ’ quoth the bard, ‘ ‘that is the question.’ ’ In order to produce the desired form, the source file must contain the sequence:

```
‘ ‘To be or not to be,’ ’ quoth the bard, ‘ ‘that is the question.’ ’
```

You are to write a program which converts text containing double-quote (") characters into text that is identical except that double-quotes have been replaced by the two-character sequences required by TEX for delimiting quotations with oriented double-quotes. The double-quote (") characters should be replaced appropriately by either ‘ ‘ if the " opens a quotation and by ’ ’ if the " closes a quotation. Notice that the question of nested quotations does not arise: The first " must be replaced by ‘ ‘, the next by ’ ’, the next by ‘ ‘, the next by ’ ’, the next by ‘ ‘, the next by ’ ’, and so on.

Input

Input will consist of several lines of text containing an even number of double-quote (") characters. Input is ended with an end-of-file character.

Output

The text must be output exactly as it was input except that:

- * the first " in each pair is replaced by two ‘ characters: ‘ ‘
- and
- * the second " in each pair is replaced by two ’ characters: ’ ’.

Sample Input

"To be or not to be," quoth the Bard, "that
is the question".
The programming contestant replied: "I must disagree.
To 'C' or not to 'C', that is The Question!"

Sample Output

‘‘To be or not to be,’’ quoth the Bard, ‘‘that
is the question’’.
The programming contestant replied: ‘‘I must disagree.
To ‘C’ or not to ‘C’, that is The Question!’’

4 Cabinets

Well-Built Cabinet Distributors, Inc. recently received an electronic catalog of cabinets from their leading manufacturer, Woodcraft. Unfortunately, the format of the data is not consistent with that expected by Well-Built's inventory software. For this problem, you will construct a program that reformats the Woodcraft catalog for use by the inventory software.

Input

Input will be formatted as from a comma-delimited ASCII file. Your program must read all input from the standard input file. Each line in the input has a maximum length of 64 characters and contains the following fields:

Field	Length	Explanation
Style Code	1-3	Code specifying the cabinet's style.
Style Name	0-15	Name for cabinet style.
Description	1-15	Code describing type of cabinet.
Extension	0-25	Additional information about cabinet.
Unit Price	0-6	Manufacturer's suggested retail price (dollars x 100).

The records are presented in ascending order by Style Code. You may assume that all fields will be consistent with the lengths given, and that all fields will contain appropriate characters.

Output

Output will consist of a reformatted catalog. The reformatted catalog file will be a comma-delimited ASCII file. The first record in the file must be the following:

Item Id,Item Desc,Item Price

Each remaining record in the file will have a maximum length of 50 characters and contain the following fields:

Field	Length	Explanation
Item Id	4-13	Unique identification code for inventory database.
Item Desc	1-30	Inventory description of cabinet.
Item Price	4-7	Manufacturer's suggested retail price.

The Item Id is formed by concatenating the Style Code and the Description. If the Style Code is less than three characters long, it must be left-filled with zeros to three characters. If the Item Id exceeds 13 characters, then the record is rejected.

The Item Desc is formed by concatenating the Style Name, a hyphen, and the Extension. If the Style Name is missing, use the Style Name from the first record of the

corresponding Style Code group. If this first record has no Style Name either, then reject the record. If the Extension is not present, then Item Desc is the same as Style Name (no hyphen). If Item Desc exceeds 30 characters, then it must be truncated on the right. The Item Price is formed by formatting the Unit Price as dollars and cents. If Unit Price is not present, then Item Price = 0.00.

Sample Input

```
23,CHAMPAGNE,BASE36,3" RECESSED TOE KICK,8900
23,,BASE54,,11000
25,LAUREL,CNR24LT,,15000
107,COLONIAL,BASE54WSIDEJAM
202,SAGEBRUSH,OVRHD54P,USE WITH HDWARE KIT #3207
221,ALVEA MODERN,BASE36
221,ALVEA MODERN,OVRHD54WCAP
```

Sample Output

```
Item Id,Item Desc,Item Price
023BASE36,CHAMPAGNE-3" RECESSED TOE KICK,89.00
023BASE54,CHAMPAGNE,110.00
025CNR24LT,LAUREL,150.00
202OVRHD54P,SAGEBRUSH-USE WITH HDWARE KIT ,0.00
221BASE36,ALVEA MODERN,0.00
```

5 TOWERS OF HANOI

Determine when the world will end

The Towers of Hanoi is a classic recursion problem:

N disks, of size N through 1, are stacked in order on the left post of three posts, so the largest disk is on the bottom.

The game is to move the stack of disks from the first post to the third post under the following constraints:

- * You may only move one disk at a time.
- * The disks on each post remain sorted from large (bottom) to small (top).

This can be accomplished recursively: to move K disks from post A to post C , first move $K-1$ disks to the remaining post, B , then move the one remaining disk from post A to post C . Moving the $K-1$ disks from post B to C is a reduced version of the original problem where the posts have been relabeled.

Legend has it that the Order of the Andes-Chilean Monks, also known as the Order of the ACM, have started the work of moving a 60-disk tower. When they finish, the world will end. Using the above recursive algorithm, this will take 1,152,921,504,606,846,975 moves.

You are to determine, to within a year, when the world will end.

INPUT

The input file will contain a sequence of starting times and rates in the format:

YYYY MM DD TMOVE

Here YYYY MM DD is the year, month, and day, using the Gregorian calendar, when the tower game began. TMOVE is the number of seconds it takes to move one disk, and is an integer value in the range $0 < TMOVE < 60$. Since the Gregorian calendar system would introduce a significant phase error over this time span, use the following convention:

- o Use Gregorian calendar conventions to convert the starting date to decimal year.
- o Use the estimate that there are 365.242199 days in a year to determine the ending date of the world.
- o Report your result using decimal year notation with one digit of precision after the decimal point.

Output

The output should be in decimal year format, with one digit of precision after the the decimal point. The answer must be correct to within a year.

Sample Input

1721 10 19 23
2001 11 17 59

Sample Output

840297140021.6
2192079493218.6

6 Sudoku Solver

Problem Description

Sudoku is a simple number puzzle invented by Leonhard Euler. The puzzle you are given is a 9x9 grid of numbers and blanks, like this:

```
+-----+-----+-----+
| _ 6 _ | 1 _ 4 | _ 5 _ |
| _ _ 8 | 3 _ 5 | 6 _ _ |
| 2 _ _ | _ _ _ | _ _ 1 |
+-----+-----+-----+
| 8 _ _ | 4 _ 7 | _ _ 6 |
| _ _ 6 | _ _ _ | 3 _ _ |
| 7 _ _ | 9 _ 1 | _ _ 4 |
+-----+-----+-----+
| 5 _ _ | _ _ _ | _ _ 2 |
| _ _ 7 | 2 _ 6 | 9 _ _ |
| _ 4 _ | 5 _ 8 | _ 7 _ |
+-----+-----+-----+
```

Note that some cells have digits already, and some are blank. The problem is solved when all blanks are filled in with the remaining digits (1 through 9 only) such that each row, column, and 3x3 box contains exactly one of each digit.

In the puzzle below, for example, none of the cells marked "x" may contain a 6. Other sixes may appear in some of the cells marked "-".

```
+-----+-----+-----+
| x 6 x | x x x | x x x |
| x x x | _ _ _ | _ _ _ |
| x x x | _ _ _ | _ _ _ |
+-----+-----+-----+
| _ x _ | _ _ _ | _ _ _ |
| _ x _ | _ _ _ | _ _ _ |
| _ x _ | _ _ _ | _ _ _ |
+-----+-----+-----+
| _ x _ | _ _ _ | _ _ _ |
| _ x _ | _ _ _ | _ _ _ |
| _ x _ | _ _ _ | _ _ _ |
+-----+-----+-----+
```

Here's the solution for the above puzzle:

```
+-----+-----+-----+
| 9 6 3 | 1 7 4 | 2 5 8 |
| 1 7 8 | 3 2 5 | 6 4 9 |
| 2 5 4 | 6 8 9 | 7 3 1 |
+-----+-----+-----+
| 8 2 1 | 4 3 7 | 5 9 6 |
| 4 9 6 | 8 5 2 | 3 1 7 |
| 7 3 5 | 9 6 1 | 8 2 4 |
+-----+-----+-----+
| 5 8 9 | 7 1 3 | 4 6 2 |
| 3 1 7 | 2 4 6 | 9 8 5 |
| 6 4 2 | 5 9 8 | 1 7 3 |
+-----+-----+-----+
```

Input and Output

Your input and output will each be a single string.

Each character of the cell is either a digit (for a pre-filled cell) or . (a blank). Each character represents one cell. The first character represents the cell $[0,0]$, the last represents $[9,9]$, the ninth the last cell of the first row, and the tenth the first cell of the second row.

For example, the example puzzle is represented as:

```
.6.1.4.5...83.56..2.....18..4.7..6..6...3..7..9.1..45.....2..72.69...4.5.8.7.
```

Your output for this input would be:

```
963174258178325649254689731821437596496852317735961824589713462317246985642598173
```

7 Problem G: Speed Limit

Bill and Ted are taking a road trip. But the odometer in their car is broken, so they don't know how many miles they have driven. Fortunately, Bill has a working stopwatch, so they can record their speed and the total time they have driven. Unfortunately, their record keeping strategy is a little odd, so they need help computing the total distance driven. You are to write a program to do this computation.

For example, if their log shows

Speed in miles per hour	Total elapsed time in hours
20	2
30	6
10	7

this means they drove 2 hours at 20 miles per hour, then $6-2=4$ hours at 30 miles per hour, then $7-6=1$ hour at 10 miles per hour. The distance driven is then $(2)(20) + (4)(30) + (1)(10) = 40 + 120 + 10 = 170$ miles. Note that the total elapsed time is always since the beginning of the trip, not since the previous entry in their log.

Input

The input consists of one or more data sets. Each set starts with a line containing an integer n , $1 \leq n \leq 10$, followed by n pairs of values, one pair per line. The first value in a pair, s , is the speed in miles per hour and the second value, t , is the total elapsed time. Both s and t are integers, $1 \leq s \leq 90$ and $1 \leq t \leq 12$. The values for t are always in strictly increasing order. A value of -1 for n signals the end of the input.

Output

For each input set, print the distance driven, followed by a space, followed by the word "miles".

Sample Input

```
3
20 2
30 6
10 7
2
60 1
30 5
4
15 1
25 2
```

30 3
10 5
-1

Sample Output

170 miles
180 miles
90 miles

8 Bipartite Numbers

The executive officers of the company where you work want to send each other encrypted messages. Rather than use off-the-shelf encryption software such as PGP, they have tasked the IT staff with handling the encryption problem. The IT staff decided on a solution that requires public and private integer keys. The idea is that everyone can see your public key, but only you know your private key.

Your best friend in the company is a wonderful person but a not-so-wonderful programmer. He has created a public-private key scheme as follows. A public key can be any positive integer. The corresponding private key is the smallest bipartite number that is greater than and a multiple of the public key.

A bipartite number is any positive integer that contains exactly 2 distinct decimal digits s and t such that s is not 0 and all occurrences of s precede all occurrences of t . For example 4444411 is bipartite (s is 4 and t is 1), So are 41, 10000000, and 5555556.

However, neither 4444114 nor 44444 are bipartite.

Notice that the large bipartite number 88888888888800000 can be nicely described as 12 8s followed by 5 0s. You can express any bipartite number using four numbers: m s n t .

The numbers s and t are the leading and trailing digits as described above, m is the number of times the digit s appears in the bipartite number, and n is the number of times the digit t appears.

The trouble with your friends scheme is that it is not too difficult to compute a private key if you know the public key. You need to convince your friend that his public-private key scheme is inadequate before he loses his job over his bad decision! You must write a program that takes public keys as input and displays the corresponding private keys.

Input

The input consists of several test cases. Each test case is on a separate line, and it consists of a single public key in the range 1 ... 99999. The last case is followed by a line containing the integer zero.

Output

For each test case, display a line consisting of the public key, a colon, then 4 integers m s n t where m , n , s , and t are as described above.

Sample Input

```
125
17502
2005
0
```

Sample Output

125: 1 5 2 0

17502: 4 7 4 8

2005: 3 2 3 5

9 Bit Compressor

The aim of data compression is to reduce redundancy in stored or communicated data. This increases effective data density and speeds up data transfer rates. One possible method to compress any binary message is the following: Replace any maximal sequence of n 1s with the binary version of n whenever it shortens the length of the message. For example, the compressed form of the data 11111110010011111111111110011 becomes 10000010011110011. The original data is 32 bits long while the compressed data is only 17 bits long. The drawback of this method is that sometimes the decompression process yields more than one result for the original message, making it impossible to obtain the original message. Write a program that determines if the original message can be obtained from the compressed data when the length of the original message (L), the number of 1s in the original message (N) and the compressed data are given. The original message will be no longer than 16 Kbytes and the compressed data will be no longer than 40 bits.

Input

The input file contains several test cases. Each test case has two lines. The first line contains L and N and the second line contains the compressed data. The last case is followed by a line containing two zeroes.

Output

For each test case, output a line containing the case number (starting with 1) and a message YES, NO or NOT UNIQUE. YES means that the original message can be obtained. NO means that the compressed data has been corrupted and the original message cannot be obtained. NOT UNIQUE means that more than one message could have been the original message. Follow the format shown in the sample output.

Sample Input

```
32 26
10000010011110011
9 7
1010101
14 14
111111
0 0
```

Sample Output

```
Case #1: YES
Case #2: NOT UNIQUE
Case #3: NO
```

10 Web Address Translation

A page on the World Wide Web may contain hypertext references, or links, to other pages on the Web. This is accomplished by insuring that each page has a unique string address (known as a URL), and each link specifies the address of the page to be visited when one clicks on the hypertext link. Web page authors are allowed to use either fully qualified addresses or a variety of shorter address forms which have some implied information. In this problem, we'll implement a program which reads a sequence of translation rules followed by a sequence of strings to be translated. For each string to be translated, the translation rules are applied in succession, potentially transforming the original string (e.g., a shortWeb address) into another string (e.g., a fully qualified Web address).

Input

The input consists of one or more nonempty lines containing translation rules, followed by an empty line, followed by one or more nonempty lines containing strings to be translated. Each translation rule has the form

```
matchtype x action y
```

where matchtype is either prefix, suffix, or substring; x and y are strings of ASCII characters not containing blanks, and action is either prepend, append, or replace. The four parts of each translation rule are separated by single blanks.

Output

For each string z to be translated, successively apply the translations rules in the order given, and output the resulting string. A prefix rule applies only if x is a prefix of z; a suffix rule applies only if x is a suffix of z; and a substring rule applies only if x is a substring z. Note that the empty string occurs as a prefix and a suffix of z, and as a substring before and after every character in z. If the action is prepend, concatenate y onto the beginning of the string; if the action is append, concatenate y onto the end of the string; if the action is replace, replace the first occurrence of x in the string with y.

Sample Input

```
prefix // prepend http:
prefix / prepend http://www.cs.wright.edu
suffix / append index.html
substring /~ replace /user/
```

```
http://www.cs.wright.edu/
/~carich/
/this/one.html
//www.cs.wright.edu/
```

Sample Output

```
http://www.cs.wright.edu/index.html  
http://www.cs.wright.edu/user/carich/index.html  
http://www.cs.wright.edu/this/one.html  
http://www.cs.wright.edu/index.html
```