2005 Mid-Atlantic Regional Programming Contest

Welcome to the 2005 Programming Contest. Before you start the contest, please be aware of the following notes:

1. There are eight (8) problems in the packet, using letters A–H. These problems are NOT sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Take Your Vitamins	Gold
B	Packing Dominoes	Orange
C	Extrusion	Purple
D	Pasting Strings	Silver
E	Mobiles	Blue
F	Context-Free Clock	Pink
G	Right-Hand Rule	Green
H	WordStack	Red

2. All solutions must read from standard input and write to standard output. In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. The judges will ignore all output sent to standard error. (You may wish to use standard error to output debugging information.) From your workstation you may test your program with an input file by redirecting input from a file:

   ```
   program < file.in
   ```

3. Solutions for problems submitted for judging are called runs. Each run will be judged. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first. DO NOT request clarifications on when a response will be returned. If you have not received a response for a run within 60 minutes of submitting it, **you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Correct	Your submission has been judged correct.
Wrong Answer	Your submission generated output that is not correct or is incomplete.
Output Format Error	Your submission's output is not in the correct format or is misspelled.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 30 seconds.

4. A team's score is based on the number of problems they solve and penalty points, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty points are charged equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty points are added for problems that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty points.

5. This problem set contains sample input and output for each problem. However, you may be assured that the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. Your major challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive an incorrect judgment, you should consider what other datasets you could design to further evaluate your program.

6. In the event that you feel a problem statement is ambiguous, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient; no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

   Additionally, you may submit a clarification request asking for the correct output for input you provide. The judges will seek to respond to these requests with the correct output. These clarification requests will be answered only when no clarifications regarding ambiguity are pending. The judges reserve the right to suspend responding to these requests during the contest. If the provided input does not meet the specifications of the problem, or contains numbers that are not representable using the available programming languages, the response "illegal input" will be returned.

   If a clarification, including output for a given input, is issued during the contest, it will be broadcast to all teams.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

8. All lines of program input and output should end with a newline character (`\n`, `endl`, or `println()`).

9. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

10. All input sets used by the judges will follow the input format specification found in the problem description.

11. Unless otherwise specified, all numbers will appear in the input and should be presented in the output beginning with the `-` if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point appears, followed by any number of decimal digits (for output of real numbers the number of decimal digits will be specified in the problem description as the "precision"). All real numbers printed to a given precision should be rounded to the nearest value.

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure you use a printing technique that rounds to the appropriate precision.

12. Good luck, and HAVE FUN!!!

Problem A: Take Your Vitamins

Manufacturers of food products are required to place nutrition information labels on their packages. A major part of this information is a listing of important vitamins and minerals, listing both the amount of the chemical present in one serving and the percentage of an adult's minimum daily requirement for that chemical.

Write a program to help prepare these nutritional labels by computing that percentage from the information on the amount present in one serving and the amount constituting the minimum daily requirement.

Input

Input consists of one or more lines, each of the form:

```
A U R V
```

where A is the amount of a vitamin/mineral present in one serving of the food product, U is the units in which A is measured, R is the minimum daily requirement for that vitamin/mineral, measured in the same units as A, and V is the name of that vitamin/mineral.

A and R will be floating point numbers. U will be a string of alphabetic characters with no embedded spaces. V will be a string of characters, possibly including spaces. A, U, R, and V will be separated from one another by exactly one space, and V is terminated by the end of the input line.

End of the input is signaled by a line in which A is negative.

Output

For each line of input data, your program should determine the percentage of the recommended daily requirement being provided for that vitamin/mineral. If it is at least 1%, your program should print a line of the form

```
V A U P%
```

where V, A, and U are the quantities from the input, and P is the percentage of the minimum daily requirement represented by the amount A.

V should be printed left-justified on the line. A should be printed with 1 digit precision, and P with zero digits precision. V, A, U, and P should be separated by one space each.

After the last such line, your program should print a line stating

```
Provides no significant amount of:
```

followed by a list of the names of all vitamins/minerals which are provided as less than 1% of the minimum daily requirement. These should be printed one name per line, in the order they occurred within the input.

Example

Input:

```
3500.0 iu 5000.0 Vitamin A
60.0 mg 60.0 Vitamin C
0.15 g 25.0 Fiber
109. mg 990. Phosphorus
0.0 mg 1000.0 Calcium
25.0 mg 20.0 Niacin
-1.0 x 0.0 x
```

Output:

```
Vitamin A 3500.0 iu 70%
Vitamin C 60.0 mg 100%
Phosphorus 109.0 mg 11%
Niacin 25.0 mg 125%
Provides no significant amount of:
Fiber
Calcium
```

Problem B: Packing Dominoes

A domino is a 2x1 oblong comprising two squares, each square marked with a set of "pips" denoting a non-negative integer. A blank square denotes the number 0 (zero).

In a standard "double-K" set of dominoes, each pair of integers in the range 0...K occurs on one domino. In other words, the dominoes are numbered:

$$(0,0),(0,1),\ldots\ (0,K),(1,1),\ldots\ (1,K),(2,2),\ldots,(K,K)$$

Given a set of dominoes and a rectangular area of width w and height h, write a program to pack that entire set of dominoes into that area subject to the limitation that, whenever two dominoes touch one another either horizontally or vertically, the numbers on the touching squares must be identical. Note that the area might not be completely covered, and, space permitting, some dominoes might not touch any others.

Input

Input will consist of one or more lines, each containing 3 positive integers. The first integer, K, indicates the size of the set of dominoes - a "double-K set" as described above. The remaining integers, w and h, indicate the size of the area in which the dominoes should be placed, measured in units the same size as one of the constituent squares of a domino. The end of input will be signaled by a non-positive K value.

Output

For each line of input, print one line containing the values K, w, and h, as defined above, followed either by a line with the string "No packing is possible" or by a report consisting of $\frac{(K+1)(K+2)}{2}$ lines, each line describing one domino, ordered as described above in the definition of a "double-K" set.

Each line of that report will contain 6 integers

$$d_1\ d_2\ x_1\ y_1\ x_2\ y_2$$

where d_1 and d_2 are the numbers on that domino, $(x1, y1)$ is the coordinates of the number d_1 in the packed layout, and (x_2, y_2) is the coordinates of number d_2. Coordinates are specified as integers with (0,0) denoting the lower-left corner of the packing area.

If multiple packings are possible, any packing meeting the criteria outlined above will be accepted.

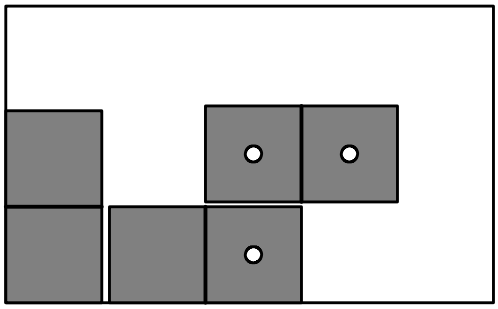
Example

Input:

```
1 5 3
3 5 2
-1 0 0
```

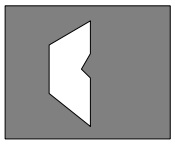
Output:

```
1 5 3
0 0 0 0 0 1
0 1 1 0 2 0
1 1 3 0 3 1
3 5 2
No packing is possible
```

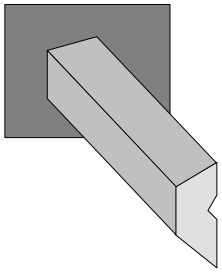


(This output corresponds to the packing shown here and is only one of many possible correct responses.)

Problem C: Extrusion



The Acme Extrusion Company specializes in the production of steel bars with custom-designed cross-sections. The manufacturing process involves cutting a hole in a thick metal plate, the shape of the hole being determined by the customer's specifications.



Molten metal is then forced through the hole to form a long bar. The shape of the hole determines the shape of the cross-section of the resulting bar.

Given a description of a polygonal hole and the volume of molten metal available, determine how long a bar can be formed by this process.

Input

Input consists of one or more data sets consisting of the following information:

- An integer, N, indicating the number of vertices making up the polygon. End of input is signaled by any N less than 3.
- Next are N lines, each containing a pair of floating-point numbers, (x_i, y_i), each denoting one vertex of the polygon. Vertices will be presented in clockwise order (relative to the closest interior point) proceeding around the perimeter of the polygon. The x_i and y_i values are in units of meters.
- The data set is terminated by a floating point value indicating the amount of molten metal available (in cubic meters).

Output

For each data set, the program should produce a single line of output of the form:

```
BAR LENGTH: x
```

where "x" is the maximum bar length, a floating point number expressed with two digits precision.

expressed as a polygon (vertices visited in clockwise order, each vertex described via (x,y) coordinates expressed in units of meters) and the volume of the available molten metal (in cubic meters),

Example

Input:

```
4
0.0 0.0
0.0 0.1
0.1 0.1
0.1 0.0
```

```
1.0
7
0.5 1.25
0.9 1.6
0.9  1.1
0.85 1.0
0.9 0.85
0.9 0.5
0.5 0.75
100.0
0
```

Output:

```
BAR LENGTH: 100.00
BAR LENGTH: 318.73
```

Problem D: Pasting Strings

You are part of a team implementing an HTML editor and have been tasked with the problem of implementing the cut/copy/paste functions. One of your goals is to preserve the formatting of selected text, even though that formatting may be determined by HTML tags outside the range of the actual selected text.

You will be provided with a block of formatted text, a starting position B, and an ending position E. Your program should output the text of the substring of that text from B (inclusive) to E (exclusive), prepending and appending formatting tags as necessary so that the output is well formed and has the same format as it had in its original position.

For the purposes of this problem, format tags consist of an opening tag (such as ""), followed by some text, followed by a closing tag (such as ""). Opening and closing tags are paired ("</*whatever*>" closes "<*whatever*>") and are considered opened between the opening tag and the closing tag. A tag may not be closed unless it is the most recent unclosed tag (e.g., "abcdefghi" is illegal). A tag may not be opened if it is already open (e.g., "recursive b" is illegal).

Input

Input data will consist of multiple test cases. Each test case will consist of one line of input of the form

```
B E TEXT
```

where B is an integer giving the (inclusive) beginning location of the substring, E is an integer giving the (exclusive) ending location of the substring, and TEXT contains the text from which to extract the substring. The TEXT begins after a single blank character immediately following E, and continues to the end of the line. B and E will be specified so that $0 \leq B \leq E \leq$ length(TEXT).

End of input will be signaled by the line "-1 -1 " with a single space following the second -1.

No input line will be longer than 200 characters.

The TEXT will be composed of characters with an ASCII value ≥ 32 (the ASCII space) and ≤ 126 (the ASCII '~'). Opening tags will be of the form "<*X*>" where *X* contains at least 1 character and is composed entirely of the characters 'a' to 'z', 'A' to 'Z', '0' to '9', and '-'. Closing tags will be of the form "</*X*>". The character '<' will only occur in the input as the first character of an opening or closing tag.

The input text will be well formed – all opening tags will be matched with a closing tag, all closing tags will match an opening tag, each closing tag will close the most recent unclosed tag, and tags will not be recursive (each tag must be closed prior to reopening).

$0 \leq B \leq E \leq$ length(TEXT). Neither B nor E will reference a character that is part of an opening or closing tag except for the character '<'.

Output

For each test case your program should print a single line containing the substring of TEXT from B (inclusive) to E (exclusive), prepending the substring with opening tags and appending the

substring with closing tags as necessary so that the output line is well formed and has the same set of open tags as when it was included in the original TEXT.

Example

Input:

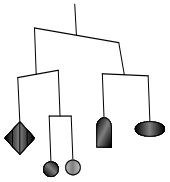
```
0 15 Testing<b>!</b>
18 23 <big>100, <bigger>1000, <biggest>10000</biggest></bigger></big>
4 4 <b>123</b>
0 16   :-/ :-> :-) :-<-> </->
-1 -1
```

Output:

```
Testing<b>!</b>
<big><bigger>1000,</bigger></big>
<b></b>
  :-/ :-> :-) :-
```

Problem E: Mobiles

Alabama Mobiles Inc. designs and manufactures *mobiles*, lightweight ”kinetic sculptures” consisting of a bar hanging from a string tied about its center. From each end of the bar hangs a string supporting either a small decorative object of some kind or another, smaller mobile. A well-designed mobile must have the weights of all the decorative items balanced, so that each bar in the mobile will naturally tend to hang horizontally. In addition, the lengths of the bars must be chosen so that each bar can rotate freely without striking the other bars, decorative objects, or the strings connecting them.



Write a program to read a partial design for a new mobile and determine if it can be balanced and if, once balanced, the elements will swing freely without bumping or entangling with one another.

Each mobile will be described as a collection of bars and decorative objects. You will be given the lengths of the bars, the weights of all but one object, and all information on how the bars and objects are connected to one another. You may assume that the bars are made of a lightweight material so that their weight and the weight of the connecting strings are negligible compared to the weights of the decorative objects. You should assume that all connecting strings are of the same length. Every mobile will have at least one bar.

Input

The input consists of a number of mobile design specifications. Each mobile design is given as a series of numbered elements, beginning with number 1 and increasing sequentially until the end of the specification.

Each element is described on one line, beginning with the the element number. The next non-blank character will be either ’B’ or ’D’ to indicate that the element is a bar or a decorative object, respectively.

If the element is a bar, the ’B’ indicator is followed by a floating-point number denoting the length of the bar, and then two integers giving the element numbers of the two elements hanging from the two ends of the bar. Every element in the input except the top bar will listed as hanging from one end of some bar.

If the element is a decorative object, the ’D’ is followed either by a single floating point number denoting the weight of that object or by the character ’X’ indicating that the weight of that object has not been determined. There will be exactly one X object in each mobile specification.

In each line describing an element, the numbers and characters making up that line are separated by one or more blank spaces.

The end of a specification is indicated by a line containing a non-positive value in the place of the element number. If that number is negative, it indicates the end of the final mobile specification in the input file.

Output

For each mobile specification, the program should print one or two lines of output. The first will either be

```
Object M must have weight NN
```

or

```
The mobile cannot be balanced.
```

where M is replaced by the element number of the "X" object and NN by the weight for that element that will balance the entire mobile (printed with two digits after the decimal point.) The second alternative should be printed if and only if no positive weight exists that will balance the mobile.

If the mobile can be balanced, the second line of output will either be

```
The mobile will swing freely.
```

or

```
The mobile will not swing freely.
```

depending upon whether, when the strings are hanging straight down, all bars are separated by a positive distance no matter how they swing around on the string.

Example

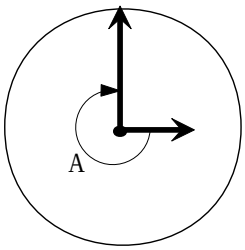
Input:

```
1 B 4.0 2 5
2 D 6.0
3 D X
4 D 3.0
5 B 3.0 3 4
-1
```

Output:

```
Object 3 must have weight 3.00
The mobile will swing freely.
```

Problem F: Context-Free Clock



You recently installed a stylish clock in your office that is perfectly round and has no markings that identify its orientation. After accidentally bumping it, you realized that 12 o'clock might no longer be at the top. Nonetheless, you want to figure out what time it is. Fortunately you recently overheard a coworker giving the time and you have a protractor and can measure the angle between the hour and minute hands.

Your program should print the first time that has the correct angle between the hour and minute hands and that is on or after the overheard time. The angle (0 to 359 degrees, inclusive) will be measured clockwise from the hour hand to the minute hand. Assume that the clock hands move smoothly.

Input

Input will consist of one test case per line, of the form

```
A HH:MM:SS
```

where A is the integral number of degrees that must be traversed clockwise to get from the hour hand to the minute hand and $HH:MM:SS$ is the overheard time in 24 hour form. $0 \leq A \leq 359$, $0 \leq HH \leq 23$, $0 \leq MM \leq 59$, and $0 \leq SS \leq 59$. HH, MM, and SS will be exactly two digits with a leading zero if necessary.

End of input will be signaled by the line

```
-1 00:00:00
```

Output

Output will consist of one line per test case, of the form

```
HH:MM:SS
```

where $HH:MM:SS$ is the first time on or after the input time where the angle from the hour hand to the minute hand is exactly A degrees, rounded *down* to the nearest second. HH, MM, and SS should be zero padded to two digits and in the same range as the input ($0 \ldots 23$, $0 \ldots 59$, and $0 \ldots 59$ respectively).

Example

Input:

```
270 14:45:00
0 12:00:00
0 12:00:01
300 13:30:00
180 08:30:00
-1 00:00:00
```

Output:

```
15:00:00
12:00:00
13:05:27
14:00:00
09:16:21
```

Problem G: Right-Hand Rule

A common approach to navigating garden mazes is to, upon entry to the maze, to place one's hand upon the wall to right of the entrance, and then to walk forward, keeping that right hand in contact with a wall at all times.

It's well known that this technique allows one to pass through one-entrance, one-exit mazes, but it does not always suffice with mazes where one is supposed to reach some goal location or locations in the interior of the maze.

Write a program to read in a description of a maze marked with a goal location and one or more entrances, and to determine whether the goal can be found by applying the right-hand rule until the goal is found or until the rule causes you to pass outside the maze through one of the entrances.

We assume that people walking through the maze will look around as they do so. Consequently, a goal is considered to have been "found" if the person steps directly onto that location or reaches any position with an unimpeded view to the goal along a vertical or horizontal line.

Input

Input consists of one or more mazes. Each maze begins with a line containing two integers, W and h, denoting the width and the height of the maze. End of input is indicated when either of these values is less than 3.

This is followed by h lines of input. In each of these lines, only the first w characters are significant. If the input line contains fewer than w characters, you should treat the missing characters as 'X'.

The interpretation of the characters in these lines is as follows:

- ' ' denotes an open space
- 'G' is an open space representing a goal location - there will be exactly one of these in any maze.
- 'X' denotes a wall
- 'E' is an open space representing an entrance. All entrances will occur on the outer perimeter (as defined by the w and h values) of the maze and no two entrances will be adjacent.

All mazes will be completely enclosed by a combination of 'X' and 'E' characters.

Output

For each maze, print a single line of output of the form

```
The goal would be found from ? out of ? entrances.
```

replacing the first '?' by the number of entrances from which the right-hand-rule allows one to find the goal and the second '?' by the total number of entrances.

Example

Input:

```
31 15
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                             X
X                             X
X                             X
X   XXXX XXXXX               X
X   X        X                X
X   X   G    X                X
X   X        X                X
X   X        X                X
X   XXXXXXXXXX                X
X                             X
X                             X
X                 XXXXXXXXXXXXX
X                             X
XXXXXXEXXXXXXXXXXXXXXXEXXXXXXX
0 0
```

Output:

```
The goal would be found from 1 out of 2 entrances.
```

Problem H: WordStack

As editor of a small-town newspaper, you know that a substantial number of your readers enjoy the daily word games that you publish, but that some are getting tired of the conventional crossword puzzles and word jumbles that you have been buying for years. You decide to try your hand at devising a new puzzle of your own.

Given a collection of N words, find an arrangement of the words that divides them among N lines, padding them with leading spaces to maximize the number of non-space characters that are the same as the character immediately above them on the preceding line. Your score for this game is that number.

Input

Input data will consist of one or more test sets.

The first line of each set will be an integer N ($1 \leq N \leq 10$) giving the number of words in the test case. The following N lines will contain the words, one word per line. Each word will be made up of the characters 'a' to 'z' and will be between 1 and 10 characters long (inclusive).

End of input will be indicated by a non-positive value for N.

Output

Your program should output a single line containing the maximum possible score for this test case, printed with no leading or trailing spaces.

Example

Input:

```
5
abc
bcd
cde
aaa
bfcde
0
```

Output:

```
8
```

One possible arrangement yielding this score is:

```
aaa
abc
 bcd
  cde
bfcde
```