Problem 1—Coast Tracker

2222 AD. The Earth Space Agency (ESA) is preparing a robot mission to Planet Atlantis, discovered just one century ago. The planet was named upon the old legend of the disappeared continent because its surface is almost entirely occupied by an immense sea. There are no continents in that planet, yet a significant part of its surface is covered by land: millions of small, unexplored islands are known to exist all over Atlantis.

One of the goals of the mission is to build a complete map of the planet's islands. As the visibility conditions are not suitable for aerial observation, ESA decided to build a set of rover robots specifically intended to make coast tracking by land. The idea is to leave one of these robots at one point in the coast of an island and let him autonomously discover the map of the entire coast, then come back as the robot returns to the same place and take it to the next island.

You are working with the team that is programming the navigation software for these robots, and some decisions were already taken: the surface will be discretised into squared, equal-sized areas; for simplification purposes, each area will be considered as being entirely covered by land or water. Also, the robot will always start its job in a square of the coast, with the sea at its right; therefore, the coast will be tracked in a counter clockwise direction (see Figure 1).

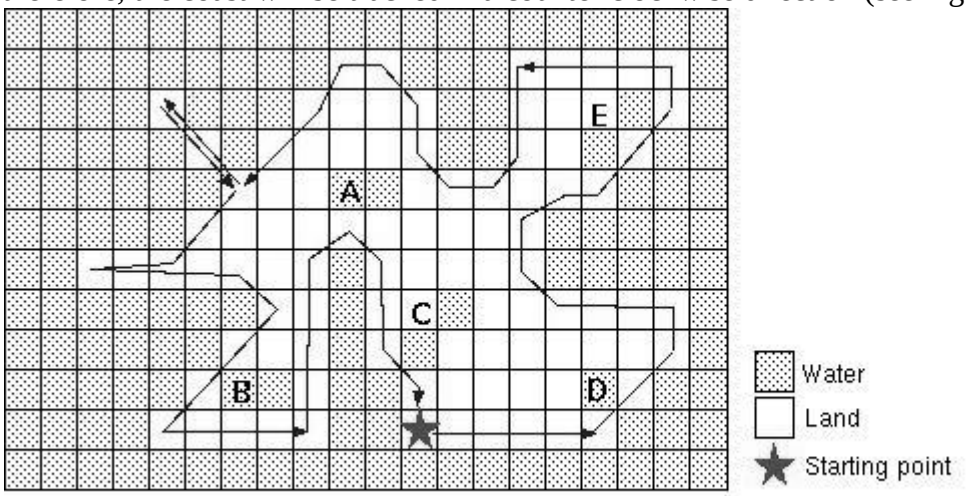


Figure 1

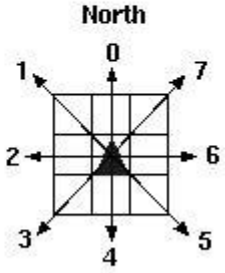


Figure 2

The robot will have a limited perception system that will be able to determine the kind of surface of the 8 areas neighbouring the one where the robot stands. Its movement capabilities will also be limited: the robot will only have the possibility of (i) rotating to one of 8 fixed directions (coded as integers from 0 to 7, 0 being North - see Figure 2) and (ii) moving to the neighbour position in front of him. Each time the robot moves to a new position, the perception system gets a new percept. There are no obstacles to worry about: God conveniently arranged the islands to have smooth, sand coasts. There are lakes (e.g., A, B, C, D and E in Figure 1), but the robot must not waste time with them: the goal is to track the seacoast only.

Given the current position and direction of the robot, and a percept of the surrounding world, decide the direction of the next move. Note that you are not being asked to program the whole coasttracking software; you are just programming a part of it. The direction must be computed in

such a way that an iterative call of the program, with percepts from an environment as the one described, would allow the robot to track the coast.
Your program must be able to process several scenarios. Each scenario comprises the robots current position and direction, and a percept. Figure 3 illustrates three hypothetical scenarios and the intended result: the direction of the next move.

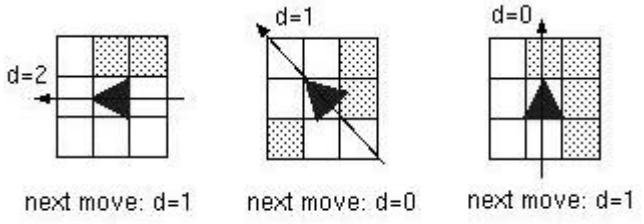


Figure 3

INPUT SPECIFICATION. The input file represents several scenarios. Input for each scenario consists of 9 lines as follows:

- First line: *x y d*, where *x* and *y* are the (always positive) coordinates of the robots current position, and *d* is the robots current direction, an integer such that 0• *d*• 7 (see Figure 2);
- Next 8 lines: x_i y_i s_i, where s_i is a number representing the surface type of the neighbouring position (x_i, y_i); land surface is represented by 1, and water surface by 0.

Successive values in a line are separated by one or more blanks. The integer -1 follows the data of the last scenario.

OUTPUT SPECIFICATION. For each given scenario, your program has to output the direction of the next robot's movement. Output for successive scenarios should be written in successive lines.

SAMPLE INPUT

```
22·25·2<EOLN>
22·26·0<EOLN>
21·26·1<EOLN>
21·25·1<EOLN>
21·24·1<EOLN>
22·24·1<EOLN>
23·24·1<EOLN>
23·25·1<EOLN>
23·26·0<EOLN>
21·26·1<EOLN>
21·27·1<EOLN>
20·27·1<EOLN>
20·26·1<EOLN>
20·25·0<EOLN>
21·25·1<EOLN>
22·25·1<EOLN>
22·26·0<EOLN>
22·27·0<EOLN>
21·27·0<EOLN>
21·28·0<EOLN>
20·28·1<EOLN>
20·27·1<EOLN>
20·26·1<EOLN>
```

```
21·26·1<EOLN>
22·26·0<EOLN>
22·27·0<EOLN>
22·28·0<EOLN>
-1<EOLN>
<EOF>
```

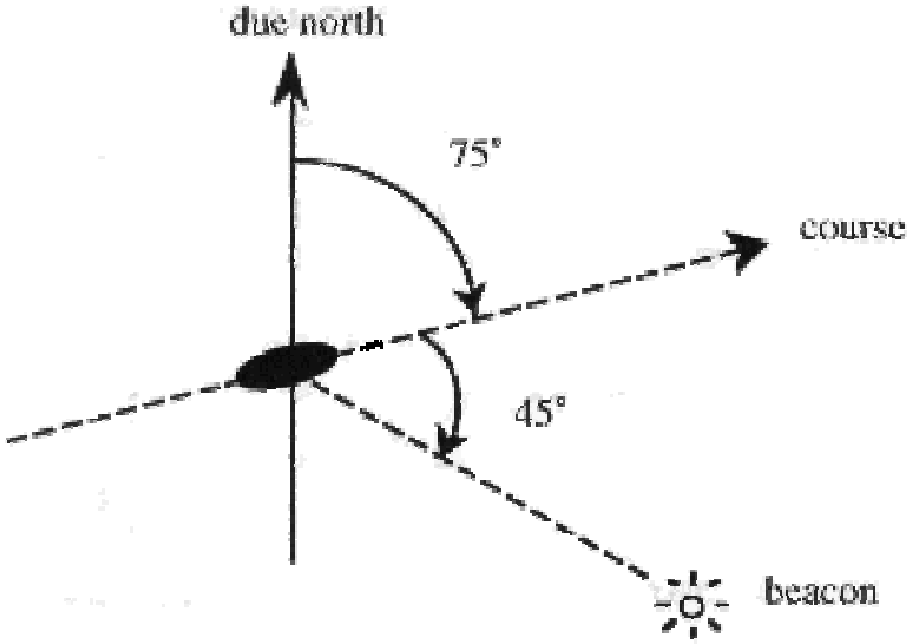
SAMPLE OUTPUT

```
1<EOLN>
0<EOLN>
1<EOLN>
<EOF>
```

Problem 2—Radio Direction Finder

A boat with a directional antenna can determine its present position with the help of readings from local beacons. Each beacon is located at a known position and emits a unique signal. When a boat detects a signal, it rotates its antenna until the signal is at maximal strength. This gives a relative bearing to the position of the beacon. Given a previous beacon reading (the time, the relative bearing, and the position of the beacon), a new beacon reading is usually sufficient to determine the boat's present position. You are to write a program to determine, when possible, boat positions from pairs of beacon readings.

For this problem, the positions of beacons and boats are relative to a rectangular coordinate system. The positive x-axis points east; the positive y-axis points north. The course is the direction of travel of the boat and is measured in degrees clockwise from north. That is, north is 0º, east is 90º, south is 180º, and west is 270º. The relative bearing of a beacon is given in degrees clockwise relative to the course of the boat. A boat's antenna cannot indicate on which side the beacon is located. A relative bearing of 90º means that the beacon is toward 90º or 270º.



The boat's course is 75º. The beacon has a relative bearing of 45º from the boat's course.

INPUT SPECIFICATION. The first line of input is an integer specifying the number of beacons (at most 30). Following that is a line for each beacon. Each of those lines begins with the beacon's name (a string of 20 or fewer alphabetic characters), the x-coordinate of its position, and the y-coordinate of its position. These fields are single-space separated.
Coming after the lines of beacon information is an integer specifying a number of boat scenarios to follow. A boat scenario consists of three lines, one for velocity and two for beacon readings.

Data on input line	Meaning of data
course speed	the boat's course, the speed at which it is traveling
time#1 name#1 angle#1	time of first beacon reading, name of first beacon relative bearing of first beacon

time#2 name#2 angle#2	time of second beacon reading, name of second beacon relative bearing of first beacon

All times are given in minutes since midnight measured over a single 24-hour period. The speed is the distance (in units matching those on the rectangular coordinate system) over time. The second line of a scenario gives the first beacon reading as the time of the reading (an integer), the name of the beacon, and the angle of the reading as measured from the boat's course. These 3 fields have single space separators. The third line gives the second beacon reading. The time for that reading will always be at least as large as the time for the first reading.

OUTPUT SPECIFICATION. For each scenario, your program should print the scenario number (Scenario 1, Scenario 2, etc.) and a message indicating the position (rounded to 2 decimal places) of the boat as of the time of the second beacon reading. If it is impossible to determine the position of the boat, the message should say ``Position cannot be determined." Sample input and corresponding correct output are shown below.

SAMPLE INPUT

```
4<EOLN>
First·2.0·4.0<EOLN>
Second·6.0·2.0<EOLN>
Third·6.0·7.0<EOLN>
Fourth·10.0·5.0<EOLN>
2<EOLN>
0.0·1.0<EOLN>
1·First·270.0<EOLN>
2·Fourth·90.0<EOLN>
116.5651·2.2361<EOLN>
4·Third·126.8699<EOLN>
5·First·319.3987<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
Scenario·1:·Position·cannot·be·determined<EOLN>
Scenario·2:·Position·is·(6.00,·5.00)<EOLN>
<EOF>
```

Problem 3—LC-Display

A friend of yours has just bought a new computer. Until now, the most powerful computer he ever used has been a pocket calculator. Now, looking at his new computer, he is a bit disappointed, because he liked the LC-display of his calculator so much. So you decide to write a program that displays numbers in an LC-display-like style on his computer.

INPUT SPECIFICATION. The input file contains several lines, one for each number to be displayed. Each line contains two integers *s*, *n* ($1 \bullet s \bullet 10$, $0 \bullet n \bullet 99999999$), where *n* is the number to be displayed and *s* is the size in which it shall be displayed.
The input file will be terminated by a line containing two zeros. This line should not be processed.

OUTPUT SPECIFICATION. Output the numbers given in the input file in an LC-display-style using *s* "-" signs for the horizontal segments and *s* "|" signs for the vertical ones. Each digit occupies exactly *s*+2 columns and 2*s*+3 rows. (Be sure to fill all the white space occupied by the digits with blanks, including the last digit.) There has to be exactly one column of blanks between two digits.

Output a blank line after each number. (You will find a sample of each digit in the sample output.)

SAMPLE INPUT

```
2·12345<EOLN>
3·67890<EOLN>
0·0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
···|····|····|·|··|·|···<EOLN>
···|····|····|·|··|·|···<EOLN>
·····<EOLN>
···|·|·····|····|····|<EOLN>
···|·|·····|····|····|<EOLN>
·····<EOLN>
<EOLN>
·····<EOLN>
|·····|·|···|·|···|·|···|<EOLN>
|·····|·|···|·|···|·|···|<EOLN>
|·····|·|···|·|···|·|···|<EOLN>
·····<EOLN>
|···|·····|·|···|·····|·|···|<EOLN>
|···|·····|·|···|·····|·|···|<EOLN>
|···|·····|·|···|·····|·|···|<EOLN>
·····<EOLN>
<EOLN>
<EOF>
```

Problem 4—Clock Angles

The angle between the hour and minute hands of an analog clock is always between 0 and 180 degrees, inclusive. For example, at 12:00 the angle between the hands is 0 degrees. At 6:00 the angle is 180 degrees, and at 3:00 the angle is 90 degrees. In this problem you are to find the angle between the hands of a clock for an arbitrary time between 12:00 and 11:59.

INPUT SPECIFICATION. The input data will contain multiple test cases. Each test case will consist of two numbers. The first number specifies the hour, and will be greater than 0 and less than or equal to 12. The second number specifies the number of minutes, and will be in the range 0 to 59. Two numbers, each of which is 0, will follow the last test case.

OUTPUT SPECIFICATION. For each test case, display the time in the usual form and the smallest angle formed by the hands, accurate to one fractional digit. The output should be similar to that shown in the sample below.

SAMPLE INPUT

```
12·0<EOLN>
12·30<EOLN>
6·0<EOLN>
3·0<EOLN>
0·0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
At·12:00·the·angle·is·0.0·degrees.<EOLN>
At·12:30·the·angle·is·165.0·degrees.<EOLN>
At·6:00·the·angle·is·180.0·degrees.<EOLN>
At·3:00·the·angle·is·90.0·degrees.<EOLN>
<EOF>
```

Problem 5—Substitution Cipher

Antique Comedians of Malidinesia would like to play a new discovered comedy of Aristophanes. Putting it on a stage should be a big surprise for the audience so all the preparations must be kept absolutely secret. The ACM director suspects one of his competitors of reading his correspondence. To prevent other companies from revealing his secret, he decided to use a substitution cipher in all the letters mentioning the new play.

A substitution cipher is defined by a substitution table assigning each character of the substitution alphabet another character of the same alphabet. The assignment is a bijection (to each character exactly one character is assigned—not necessarily different). The director is afraid of disclosing the substitution table and therefore he changes it frequently. After each change he chooses a few words from a dictionary at random, encrypts them and sends them together with an encrypted message. The plain (i.e. non-encrypted) words are sent by a secure channel, not by mail. The recipient of the message can then compare plain and encrypted words and create a new substitution table.

Unfortunately, one of the ACM cipher specialists have found that this system is sometimes insecure. Some messages can be decrypted by the rival company even without knowing the plain words. The reason is that when the director chooses the words from the dictionary and encrypts them, he never changes their order (the words in the dictionary are in alphabetical order). The director is interested in which of his messages could be read by the rival company. You are to write a program to determine that.

INPUT SPECIFICATION. The input consists of N cases. The first line of the input contains only positive integer N. Then follow the cases. The first line of each case contains only two positive integers A, $1 \bullet A \bullet 26$, and K, $1 \bullet K \bullet 20$, separated by a space. A determines the size of the substitution alphabet (the substitution alphabet consists of the first A lowercase letters of the english alphabet (a-z) and K is the number of encrypted words. The plain words contain only the letters of the substitution alphabet. The plain message can contain any symbol, but only the letters of the substitution alphabet are encrypted. Then follow K lines, each containing exactly one encrypted word no longer than 20 characters each. The next line contains the encrypted message containing no more than 50 total characters.

OUTPUT SPECIFICATION. For each case, print exactly one line. If it is possible to decrypt the message uniquely, print the decrypted message. Otherwise, print the sentence "Message cannot be decrypted."

SAMPLE INPUT

```
2<EOLN>
5·6<EOLN>
cebdbac<EOLN>
cac<EOLN>
ecd<EOLN>
dca<EOLN>
aba<EOLN>
bac<EOLN>
cedab<EOLN>
```

4·4<EOLN>
cca<EOLN>
cad<EOLN>
aac<EOLN>
bca<EOLN>
bdac<EOLN>
<EOF>

SAMPLE OUTPUT

abcde<EOLN>
Message·cannot·be·decrypted.<EOLN>
<EOF>

Problem 6—Flooded!

To enable homebuyers to estimate the cost of flood insurance, a real-estate firm provides clients with the elevation of each 10-meter by 10-meter square of land in regions where homes may be purchased. Water from rain, melting snow, and burst water mains will collect first in those squares with the lowest elevations, since water from squares of higher elevation will run downhill. For simplicity, we also assume that storm sewers enable water from high-elevation squares in valleys (completely enclosed by still higher elevation squares) to drain to lower elevation squares, and that water will not be absorbed by the land.

From weather data archives, we know the typical volume of water that collects in a region. As prospective homebuyers, we wish to know the elevation of the water after it has collected in low-lying squares, and also the percentage of the region's area that is completely submerged (that is, the percentage of 10-meter squares whose elevation is strictly less than the water level). You are to write the program that provides these results.

INPUT SPECIFICATION. The input consists of a sequence of region descriptions. Each begins with a pair of integers, m and n, each less than 30, giving the dimensions of the rectangular region in 10-meter units. Immediately following are m lines of n integers giving the elevations of the squares in row-major order. Elevations are given in meters, with positive and negative numbers representing elevations above and below sea level, respectively. The final value in each region description is an integer that indicates the number of cubic meters of water that will collect in the region. A pair of zeroes follows the description of the last region.

OUTPUT SPECIFICATION. For each region, display the region number (1, 2, ...), the water level (in meters above or below sea level) and the percentage of the region's area under water, each on a separate line. The water level and percentage of the region's area under water are to be displayed accurate to two fractional digits. Follow the output for each region with a blank line.

SAMPLE INPUT

```
3·3<EOLN>
25·37·45<EOLN>
51·12·34<EOLN>
94·83·27<EOLN>
10000<EOLN>
0·0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
Region·1<EOLN>
Water·level·is·46.67·meters.<EOLN>
66.67·percent·of·the·region·is·under·water.<EOLN>
<EOLN>
<EOF>
```