ACM North Central Regional Programming Contest

November 2000 Problem Statements

The problem statements presented here are in PDF format and represent unmodified versions of the originals used in the contest. Any potential clarifications to the problem statements are available elsewhere. The "tiebreaker" problem was not used, and is therefore not published here.

Problem	Title
1	An Inductively-Defined Function
2	The Mobius Function
3	Ro and Bot Meet
4	Minesweeper
5	Before and After
6	The Spiral of Primes
7	Maze Checking and Visualization
8	Recognizing S Expressions

2000-2001 ACM North Central Regional Programming Contest
Problem 1 — An Inductively-Defined Function

Consider the function *f* which is inductively defined on the positive integers, as follows:

$f(1) = 1$

$f(2n) = n$

$f(2n+1) = f(n) + f(n+1)$

Given a positive integer value for *n* (greater than or equal to 1), find the value of $f(n)$.

Input

The input consists of a sequence of positive integer values for *n* followed by –1. The integers are preceded and/or followed by whitespace (blanks, tabs, and ends of lines).

Output

For each positive integer *n*, display the value of *n* and the value of $f(n)$. Use the format shown in the example below, and leave a blank line between the output for each value of *n*.

Sample Input

```
2    53

        153
-1
```

Expected Output

```
f(2) = 1

f(53) = 27

f(153) = 77
```

2000-2001 ACM North Central Regional Programming Contest
Problem 2 — The Mobius Function

The Mobius function $M(n)$ is defined on positive integers as follows:

$M(n) = 1$ if n is 1.

$M(n) = 0$ if any prime factor of n is contained in n more than once.

$M(n) = (-1)^p$ if n is the product of p different prime factors.

For example:

M(78) = –1, since $78 = 2 \times 3 \times 13$.

M(34) = 1, since $34 = 2 \times 17$.

M(45) = 0, since $45 = 3 \times 3 \times 5$.

Given a value for n greater than or equal to 1 and less than or equal to 10000, find the value of $M(n)$.

Input

The input consists of a sequence of positive integer values for n followed by –1. The integers are preceded and/or followed by whitespace (blanks, tabs, and ends of lines).

Output

For each positive integer n, display the value of n and the value of $f(n)$. Use the format shown in the example below, and leave a blank line between the output for each value of n.

Sample Input

```
78
   34        45
       105
   1
          -1
```

Expected Output

```
M(78) = -1

M(34) = 1

M(45) = 0

M(105) = -1

M(1) = 1
```

2000-2001 ACM North Central Regional Programming Contest
Problem 3 — Ro and Bot Meet

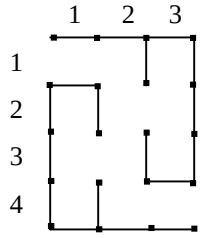
A flat rectangular maze is constructed from square cells. Each cell has one, two, or three open sides. The maze has two entry points from outside the maze, one in the cell at the upper left corner of the maze, and the other in the cell at the lower right corner of the maze. A maze may have dead ends, but it does not contain any loops. That is, there is no forward path through a maze that will lead back to a previously visited cell.

At the same time, two robots (named Ro and Bot) enter and begin traversing the maze. Ro enters at the upper left corner, and Bot enters at the lower right corner. Each robot takes the same amount of time to move from one cell to the next. The robots follow these rules in traversing the maze:

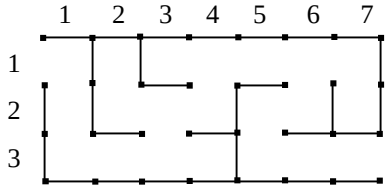
- If a robot enters a cell that has only one open side (a dead end), the robot turns around and leaves the cell.
- If a robot enters a cell that has two open sides, it leaves using the side through which it did not enter.
- If a robot enters a cell that has three open sides *A*, *B* and *C* (labeled in clockwise order), and was <u>first</u> entered through side *A*, the robot must choose to leave through side *B* or side *C*. Ro will choose to exit through side *B*, and Bot will choose to exit through side *C*. If a dead end later causes a robot to return to the cell, it will then leave through the last remaining open side. For example, if Ro enters a cell with three open sides through side *A*, it will leave through side *B*. If that route leads to dead ends, Ro eventually returns to the cell and leaves through side *C*. Naturally if all paths reached through sides *B* and *C* lead to dead ends, the robot eventually retreats back through side A (which it first used when it entered the cell).
- The robots stop if they meet in a cell (at the same time) or if they exit the maze.

You are to write a program that will determine from the description of a maze if the robots will stop inside the maze, and if they do, the cell at which they will stop.

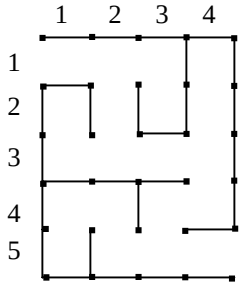
A few examples will clarify these ideas. Shown below are three mazes. Below each maze is a table showing the cells through which the robots will pass while traversing the maze. Row and column numbers are used to identify the cells in each maze.



Ro		Bot	
Row	*Col*	*Row*	*Col*
1	1	4	3
1	2	4	2
2	2	3	2
2	3	2	2
1	3	2	3
2	3	3	3
3	3	2	3
2	3	1	3
2	2	2	3
3	2	2	2
4	2	1	2
4	3	1	1
exits		*exits*	



Ro		Bot	
Row	*Col*	*Row*	*Col*
1	1	3	7
2	1	3	6
3	1	3	5
3	2	2	5
3	3	2	6
2	3	1	6
2	2	1	7
1	2	2	7
2	2	1	7
2	3	1	6
2	4	1	5
1	4	1	4



Ro		Bot	
Row	*Col*	*Row*	*Col*
1	1	5	4
1	2	5	3
1	3	4	3
2	3	4	4
1	3	3	4
1	2	2	4
2	2	1	4
3	2	2	4
3	3	3	4
3	4	3	3
2	4	3	2
1	4	2	2
2	4	1	2
3	4	1	3
4	4	2	3
4	3	1	3
5	3	1	2
5	4	1	1
exits		*exits*	

Input

The input data will contain descriptions of multiple mazes. The description of each maze begins with integers giving the number of rows (*NR*) and number of columns (*NC*) in the maze. Neither *NR* nor *NC* will be larger than 20. Following these integers there will appear *NR* × *NC* hexadecimal digits, corresponding to the cells in the maze in row-major order. Blanks and end of line characters may be included at arbitrary places for readability. Hexadecimal digits include the decimal digits **0** through **9** (representing themselves), and the upper case letters **A** through **F** (representing the values 10 through 15 respectively). Each hexadecimal digit identifies the open sides of the corresponding cell in the maze, as follows. Each side of a cell has an associated number: top = 1, right = 2, bottom = 4, and left = 8. If the numbers corresponding to the open sides of a cell are totaled, they yield the corresponding hexadecimal digit that will appear in the input for that cell. For example, a cell that has only its left and right sides open would be specified in the input as the hexadecimal digit **A**, since 8 (left) + 2 (right) = 10. A cell with its right, bottom, and left sides open would be specified in the input as the hexadecimal digit **E**, since the value of **E** is 14, or 2 (right) + 4 (bottom) + 8 (left).

A pair of zeroes follows the data for the last case.

Output

The output for each maze must begin with the word **Maze** followed by the maze sequence number (they are numbered sequentially starting with 1), a colon, and a blank. This is then followed by the message

The robots do not meet.

or the message

The robots meet in row *R*, column *C*.

as appropriate (with ***R*** and ***C*** replaced by the row and column number at which the robots meet). Leave a single blank line between the output for each maze.

Sample Input

```
4 3 A C 4 4 7 D 7 D 1 1 3 A
3 7 C 4 2 E A E C 5 3 E 9 6 9 1 3 A B 8 3 A A
5 4 A E C 4 4 5 1 5 3 B A D 6 C 6 9 1 3 B A
0 0
```

Expected Output

```
Maze 1: The robots do not meet.

Maze 2: The robots meet in row 1, column 4.

Maze 3: The robots do not meet.
```

2000-2001 ACM North Central Regional Programming Contest
Problem 4 — Minesweeper

The single player game of minesweeper is played on a rectangular grid. Each cell in this grid is either vacant or contains a mine. When the game begins, all cells are covered so the player cannot tell if a cell contains a mine. Plays consist of selecting and uncovering cells, one at a time. If an uncovered cell contains a mine, the game is over. If the cell does not contain a mine, it reveals an integer that indicates the number of neighboring cells that do contain mines. Neighboring cells are those that are immediately adjacent either horizontally, vertically, or diagonally.

In this problem, you are given a game grid on which some number of plays has already been made without uncovering a mine. You are to determine the status of those still covered cells whose status can be determined using only the information revealed by the uncovered cells. That is, for each cell still covered, determine if its status can be discovered, and if so, whether or not it contains a mine.

As an example, consider the grids shown below. The cells with the darker background color and without a number represent cells that are still covered. The cells containing a number are uncovered, and the number each contains indicates the number of neighboring cells that contain mines. In the left grid, it is clear that the rightmost uncovered cell in the top row contains a mine, as do the last uncovered cell in the leftmost column and the rightmost uncovered cell in the second row from the top. These cells have been marked with an asterisk (*) for clarity. We can now determine that the covered cells marked with a plus sign (+) have no mines. The status of the cell in the upper left corner of the grid cannot be determined. In the right grid, we cannot unambiguously determine the status of any uncovered cells. It could be that the first or second cell in the top row contains a mine, with the other cell not containing a mine.

	+	+	*	1
+	*	2	1	1
+	2	1	0	0
*	1	0	0	0
1	1	0	0	0

1	1

Input

The input data will contain multiple cases. Each case begins with integers giving the number of rows and columns in the grid. There will be no more than 10 rows and 10 columns in the grid. Following these there will be one integer for each cell in the grid, given in row major order. For a covered cell, the value –1 is given. For an uncovered cell, the number of neighboring (covered) cells that contain mines is given, a value between 0 and 8. No uncovered cells contain mines.

A pair of zeroes follows the data for the last case.

Output

For each case, first display the case number. Cases are numbered sequentially, starting with 1. Then display a list with the row and column number of each cell that definitely contains a mine, and another list with the row and column number of each cell that definitely does not contain a mine. The lists are to be displayed in a format similar to that shown in the sample below. Separate the output for consecutive cases with a blank line.

Sample Input

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

2 2
-1 -1
 1  1

0 0
```

Expected Output

```
Case 1:
    Cells with mines: (1,4), (2,2), (4,1)
    Cells without mines: (1,2), (1,3), (2,1), (3,1)

Case 2:
    Cells with mines:
    Cells without mines:
```

2000-2001 ACM North Central Regional Programming Contest
Problem 5 — Before and After

A popular television game show requires contestants to identify a phrase, usually with some of the letters missing. Some of these phrases are formed from a combination of two phrases that are unrelated, except that the last word (or words) of the first phrase are the same as the first word (or words) of the second phrase.

For example, consider the phrases “the shallow end of the pool” and “the pool table.” The last two words of the first phrase (“the pool”) are the same as the first two words of the second phrase. These phrases could then be combined to yield the “before and after” phrase “the shallow end of the pool table.”

In this problem you are given a set of phrases and asked to find all possible “before and after” phrases that can be produced. For each such combination, you are to display the two phrases, one per line, with the common words aligned.

Input

The input data contains one line for each phrase, with a blank line following the last phrase. Each phrase will consist of words containing only lower-case alphabetic characters. Words are separated by one or more blanks, and blanks may precede or follow the phrase. No word will contain more than 12 letters, and no input line will contain more than 50 characters.

Output

Display three lines for each “before and after” phrase that can be produced from the phrases given in the input. On the first line, display the first phrase starting in column 1. On the second line, display the second phrase indented so the overlapping word (or words) are perfectly aligned under the matching words in the first phrase. The third line is blank. Each phrase should be displayed with exactly one blank between adjacent words.

The order in which the “before and after” phrases are displayed should correspond to the order in which the first phrase of the combination appears in the input data.

Sample Input

```
house  of  cards
     the pool table
  the white house
      the house limit
the  shallow  end  of  the  pool
```
This line is blank.

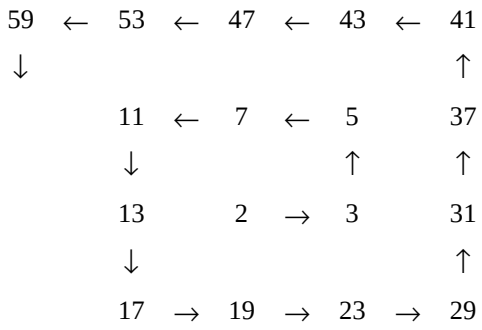
Expected Output

```
the white house
          house of cards

the shallow end of the pool
                  the pool table
```

2000-2001 ACM North Central Regional Programming Contest
Problem 6 — The Spiral of Primes

Prime numbers (those that have only themselves and other primes as factors) starting with 2 can be arranged on a two-dimensional plane starting as show below:



Assume a Cartesian coordinate system is used to reference the primes, and that 2 is located at (0,0). We can then see that 3 is located at (1,0), 5 is located at (1,1), 7 is located at (0,1) and 11 is located at (–1,1). Given the coordinates of a prime number in this system, find and display the prime number at that location.

Input

The input data will contain multiple pairs of integers, each pair representing the coordinates of a prime number. The last pair will be followed by a single integer –999. No prime will be larger than 10000.

Output

For each coordinate pair, display the input case number (starting with 1), the coordinate pair, and the prime at that coordinate location. Display one blank line between the output for each case. Your output should resemble the format shown below.

Sample Input

```
1 1
2 2
-2 2
-999
```

Expected Output

```
Case 1: The prime at location (1,1) is 5.

Case 2: The prime at location (2,2) is 41.

Case 3: The prime at location (-2,2) is 59.
```

2000-2001 ACM North Central Regional Programming Contest
Problem 7 — Maze Checking and Visualization

A program is needed to assist in checking and visualizing the mazes for problem 3. In that problem, the input consists of a sequence of maze descriptions. The mazes are rectangular, and consist of a number of square cells. Some of the sides of these cells are solid, and other sides are open.

To be consistent, an open side in one cell must correspond to an open side on the adjacent neighboring cell. The maze characteristics also require that there be only two entry points to the maze, one on the left side of upper left cell in the maze, and one on the right side of the lower right cell in the maze.

These maze characteristics can be easily verified if a visual representation of the maze is available. To prepare such a representation, each cell in the maze is approximately displayed using plus signs at the corners of cells, three hyphens (or minus signs) for a solid top or bottom side, and a single vertical stroke for a solid left or right side. An appropriate number of blanks should be used for any open sides of a cell. If a side shared by two cells is inconsistent (solid in one cell and open in the other), then print three lower-case x's in place of the three hyphens if a top or bottom side is inconsistent, or a single upper case X in place of the vertical stroke if a left or right side is inconsistent.

A cell with solid sides is shown on the left below. To its right a two row, two-column maze is shown. This maze has an open side to the left of the upper left cell and to the right of the lower right cell. The visualization also shows an inconsistency between the right side of the upper left cell and the left side of the upper right cell, and an inconsistency between the lower side of the upper right cell and the upper side of the lower right cell.

```
+---+           +---+---+
|   |               X   |
+---+           +   +xxx+
                |   |
                +---+---+
```

Display a visual representation of each maze in the input data.

Input

The input data format for this program is identical to that of problem 3.

The input data will contain descriptions of multiple mazes. The description of each maze begins with integers giving the number of rows (*NR*) and number of columns (*NC*) in the maze. Neither *NR* nor *NC* will be larger than 20. Following these integers there will appear *NR* × *NC* hexadecimal digits, corresponding to the cells in the maze in row-major order. Blanks and end of line characters may be included at arbitrary places for readability. Hexadecimal digits include the decimal digits **0** through **9** (representing themselves), and the upper case letters **A** through **F** (representing the values 10 through 15 respectively). Each hexadecimal digit identifies the open sides of the corresponding cell in the maze, as follows. Each side of a cell has an associated number: top = 1, right = 2, bottom = 4, and left = 8. If the numbers corresponding to the open sides of a cell are totaled, they yield the corresponding hexadecimal digit that will appear in the input for that cell. For example, a cell that has only its left and right sides open would be specified in the input as the hexadecimal digit **A**, since 8 (left) + 2 (right) = 10. A cell with its right, bottom, and left sides open would be specified in the input as the hexadecimal digit **E**, since the value of **E** is 14, or 2 (right) + 4 (bottom) + 8 (left).

A pair of zeroes follows the data for the last case.

Output

The output for each maze will begin with identification of the maze number. Mazes are numbered sequentially starting with 1. Following this identification there should be a single blank line and the visualization of the maze as previously described. The maze should be displayed starting in column one of each output line. A blank line should follow each maze.

Sample Input

```
4 3 A C 4 4 7 D 7 D 1 1 3 A
3 7 C 4 2 E A E C 5 3 E 9 6 9 1 3 A B 8 3 A A
5 4 A E C 4 4 5 1 5 3 B A D 6 C 6 9 1 3 B A
0 0
```

Expected Output

```
Maze 1:

+---+---+---+
        |   |
+---+   +   +
|   |       |
+   +xxx+   +
|       X   |
+   +   +xxx+
|   |
+---+---+---+

Maze 2:

+---+---+---+---+---+---+---+---+
    |   |   X                   |
+   +   +---+xxx+---+   +   +   +
|   |           |       |   |   |
+   +---+   +---+   +---+---+---+
|               |
+---+---+---+---+---+---+---+---+

Maze 3:

+---+---+---+---+---+
            |   |
+---+   +   +   +   +
|   |   |   |   |   |
+xxx+   +---+   +   +
|               |   |
+---+---+---+   +   +
|       |       |   |
+   +   +   +---+   +
|   |                |
+---+---+---+---+---+
```

2000-2001 ACM North Central Regional Programming Contest
Problem 8 — Recognizing S Expressions

The functional programming language LISP uses *s-expressions* to represent programs and the data on which they operate. A simplified definition of an s-expression is as follows.

1. Any single alphabetic letter, upper or lower case, is an s-expression.
2. If u and v are s-expressions, then so is **(**u**,**v**)** .

For example, each of the letters **a**, **c** and **m** is individually a legal s-expression, as confirmed by part 1 of the definition. Using part 2 of the definition, we determine that **(a,c)** is also a legal s-expression, and then so is **((a,c),m)**.

Your task, in this problem, is to develop a recognizer for s-expressions of the form just described. For each potential s-expression given in the input, you are to determine if it is legal and to display your finding.

Input

The input data will contain multiple candidates for s-expressions. Each line of the input will contain an s-expression candidate. There may be blanks (spaces) before, after, or both before and after the candidate. No input line will contain more than 72 characters. A blank line (that is, a line containing only zero or more blanks) will follow the last candidate.

Output

For each candidate, display the input line number (starting with 1), the candidate (without any leading blanks), and an indication of whether it is an s-expression or not. Display one blank line between the output for each candidate. Your output should resemble the format shown below.

Sample Input

```
a
  b
    (a,b)
((a,b),(C,d))
  (ab,c)
     [ a , b ]
```
This line is blank.

Expected Output

```
1: a
   is an s-expression.

2: b
   is an s-expression.

3: (a,b)
   is an s-expression.

4: ((a,b),(C,d))
   is an s-expression.

5: (ab,c)
   is not an s-expression.

6: [ a , b ]
   is not an s-expression.
```