2002 Mid-Atlantic Regional Programming Contest
Sponsored by IBM

Welcome to the 2002 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 necessarily 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	A Simple Question of Chemistry	Blue
B	An Amazing Problem	Green
C	Spelling Be	Pink
D	Apply a Cold Compress	Red
E	A Baron Landscape	Gold
F	In Defence of a Garden	Purple
G	Cantoring Along	Silver
H	Chambers Ceramic Conundrum	Orange

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. 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 90 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 issue a clarification request about a submission you have not received a judgment for.**

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.
Incorrect Output	Your submission generated output that is not correct.
Incorrect Output Format	Your submission's output is not in the correct format.
Incomplete Output	Your submission failed to generate all required output.
Syntax Error	Your submission failed to compile on the judge's machine.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission failed to complete within 30 seconds.

4. 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 do not believe that you have discovered an ambiguity in the problem, you will receive the response, "The problem statement is not ambiguous, 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 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 a clarification, including output for a given input, is issued during the contest, it will be broadcast to all teams.

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

6. Good luck, and HAVE FUN!!!

Problem A: A Simple Question of Chemistry

Your chemistry lab instructor is a very enthusiastic graduate student who clearly has forgotten what their undergraduate Chemistry 101 lab experience was like. Your instructor has come up with the brilliant idea that you will monitor the temperature of your mixture every minute for the entire lab. You will then plot the rate of change for the entire duration of the lab.

Being a promising computer scientist, you know you can automate part of this procedure, so you are writing a program you can run on your laptop during chemistry labs. (Laptops are only occasionally dissolved by the chemicals used in such labs.) You will write a program that will let you enter in each temperature as you observe it. The program will then calculate the difference between this temperature and the previous one, and print out the difference. Then you can feed this input into a simple graphing program and finish your plot before you leave the chemistry lab.

Input

The input is a series of temperatures, one per line, ranging from -10 to 200. The temperatures may be specified up to two decimal places. After the final observation, the number 999 will indicate the end of the input data stream. All data sets will have at least two temperature observations.

Output

Your program should output a series of differences between each temperature and the previous temperature. There is one fewer difference observed than the number of temperature observations (output nothing for the first temperature). Differences are always output to two decimal points, with no leading zeroes (except for the ones place for a number less than 1, such as 0.01) or spaces.

After the final output, print a line with “End of Output”

Example

Input:

```
10.0
12.05
30.25
20
999
```

Output:

```
2.05
18.20
-10.25
End of Output
```

Problem B: An Amazing Problem

The recent improvements in robotics have allowed archaeologists to explore previously unknown areas of the pyramids in Egypt. You have been commissioned to write a program to help one of these robots navigate. To deal with some of the winding passages and the thick stone that absorbs radio waves, the robot will need its own autonomous navigation. You must develop a way for the robot to navigate itself out of mazes it may find itself in using only its vision of nearby walls.

Your solution will be a program that selects a path to navigate the robot out of a maze. The challenging part of this problem is that your program must be interactive. At each step the program will be given a description of the spaces surrounding the robot. Your program must then decide the direction in which to move, send the command to the robot, and read in the description of the spaces around the robot after the move.

For convenience, the maze is a Cartesian grid. Each space on the grid is empty space (‘ ’), a stone wall (`X`), or a space outside the maze (`O`) (the letter, not the digit). The description of the space your robot receives will be a 3x3 grid, with your robot in the center denoted by a `+`. The maze is at most 100×100 squares.

(Problem continued on following page)

Example

As an example, consider the following maze that your robot is currently in (your robot will not receive this as input):

```
0000000000
0XXXXXXXX0
0XXXX XXX0
0       +X0
0XXXX XXX0
0XXXXXXXX0
0000000000
```

The first input your program receives is the area surrounding its starting location:

```
XXX
 +X
XXX
```

Your program issues a direction to move, by outputting one of `N`, `S`, `E`, or `W` followed by a newline. You may only move in one of these four directions. Your program outputs its movement and then receives the next set of surrounding squares (your program's output is indicated by `>` and its input indicated by `<`. You should not expect or generate these characters. They are for illustration in this example only.)

```
>W
< XX
< +
< XX
>W
<X X
< +
<X X
```

Your program must continue until it issues the instruction that moves the robot onto an `O` square. There will be a valid path from your robot's start position to an O square. An invalid move into a wall (`X`) will result in an "Incorrect Output" judgment.

Your robot may be started at any empty position inside the maze.

Problem C: Spelling Be

It's a simple requirement your company has, really—every document should be spell-checked before it's sent out to a customer. Unfortunately, while word processing documents are easily spell-checked, your employees have not been checking email every time they send out a message. So you've come up with a little improvement. You are going to write a program that will check email on its way out. You will spell-check each message, and if you find any spelling errors, it will be returned to the sender for correction.

When you announced this plan, one of your coworkers fell off their chair laughing, saying that you couldn't possibly anticipate every name, technical acronym, and other terms that might appear in an email. Undaunted, however, you are going to test-run your code with an online dictionary and some sample emails you have collected.

Input

The input consists of two sections, the dictionary and the emails. The first line of input specifies the number of words in the dictionary, followed by that many lines, with one word per line. There is no whitespace before, after, or in any words, although there may be apostrophes or hyphens in the words, which are considered part of the word (i.e. "bobs" is different than "bob's". There will be no duplicate words. All words will be in lower case.

Following that are the emails. The first line of this section has the number of emails in the input. Following that line begins the first email. It has been preprocessed, so it consists of one word per line, with no punctuation (other than apostrophes and hyphens) or whitespace, and all words are in lower case. The last word in the email is followed by a line with only "-1". Each email will have at least one word.

Output

For each email, you must either print:

```
Email X is spelled correctly.
```

where X begins with 1 and counts up. Or, if a word is found that is not in the dictionary, print out:

```
Email X is not spelled correctly.
```

followed by a list of unknown words in the order that you find them, one per line. If an unknown word is found multiple times, it should be printed multiple times.

There are no spaces between datasets. Following the output for the final dataset, print a line stating "End of Output"

Example

Input:

```
6
alice
am
bitterly
i
leaving
you
1
dear
bob
i
am
leaving
you
bitterly
alice
-1
```

Produces output:

```
Email 1 is not spelled correctly.
dear
bob
End of Output
```

Problem D: Apply a Cold Compress

Many techniques for compressing digital graphics focus on identifying and describing regions of a single uniform character. Here is a simple technique for compressing black-and-white images (which could be easily extended to color). The basic idea is to repeatedly split the original picture in half, either vertically or horizontally, until each of the resulting sub-pictures contains only a single color.

A rectangular digital graphic is described by a "compression-expression," defined as follows:

Each compression-expression begins with a two-bit tag, which may be followed by additional compression-expressions depending upon the tag value. The tag values are interpreted as follows:

00 A square region that consists entirely of black pixels. This region may be a single pixel, a 2x2 square, a 3x3 square, etc., depending upon context.

11 A square region that consists entirely of white pixels. This region may be a single pixel, a 2x2 square, a 3x3 square, etc., depending upon context.

10 A horizontal split. This is followed by two compression expressions. The picture produced by a split is formed by taking the pictures denoted by each of those two expressions and placing them along-side one another, the first picture to the left and the second to the right.

Horizontal splits are only possible between two pictures of the same height.

01 A vertical split. This is followed by two compression expressions. The picture produced by a split is formed by taking the pictures denoted by each of those two expressions and placing them along-size one another, the first picture on the top and the second underneath it.

Vertical splits are only possible between two pictures of the same width.

When interpreting splits, it may be necessary to change the scale of the components to make them compatible. For example, given a 2:6 picture A (i.e., 2 pixels wide, 6 pixels high) and a 3:4 picture B:

- A vertical split involving these two is possible only if we scale A by a factor of 3, making it 6:18, and scale B by a factor of 2, making it 6:8. The resulting combined picture would have size 6:26.
- A horizontal split involving these two is possible only if we scale A by a factor of 2, making it 4:12, and scale B by a factor of 3, making it 9:12. The resulting combined picture would have size 13:12.

For example, using Xs and ' 's to denote black and white pixels, respectively, the expression "00" denotes the picture

```
---
|X|
---
```

and the expression “1000010011” denotes

```
-----
|XXX|
|XX |
-----
```

Examination of this format will show that for any given compression-expression, there is some smallest picture that can be denoted by that expression, but the same expression can also denote pictures twice the size of the smallest one, three times the size, etc.

Input

Each line of the input will contain a compression-expression, presented as a single line containing an arbitrary number of 0’s and 1’s. The input ends following the line with the final compression-expression.

All input sets used in this problem will be valid compression-expressions.

Output

For each line of input, your should print the smallest black-and-white picture denoted by that expression, drawn in `X`s (black) and ‘ ’s (white), as above, and framed in `-` and `|` characters as shown in the examples. There should be no characters or whitespace outside your frame except for the newlines terminating each line.

There should be no blank lines in your output.

Example

Given the input

```
00
1000101110010010111011110111000100011
```

the output should be

```
---
|X|
---
----------------
|XXXX    XXX   |
|XXXX    XXX   |
|XXXX    XXX   |
|XXXX       XX |
----------------
```

Problem E: A Baron Landscape

After a successful military campaign, the King decided to reward his two most able commanders with a title and a portion of the newly conquered territory. Each of the newly appointed Barons will be allowed to construct a castle in the new territory and to collect taxes from the surrounding lands.

The King has commissioned a map of the new territory, marked off in a grid. Each square on the grid is approximately the distance a man on horseback could ride in one day. Each Baron will choose a square in which to build his castle. As the senior commander, you will choose first.

For the sake of this selection, castles will be presumed to lie at the center of the selected square. The two castles must be built in squares whose centers are more than three days' ride from one another.

Each Baron will be allowed to collect taxes from the peasants in any square whose center is 6 days' ride or less from that Baron's castle and that is closer to that Baron's castle than to the other Baron's. Squares that are equidistant from the two castles do not contribute taxes to either castle.

Tensions had been rising between you and your fellow commander throughout the military campaign. You are certain that, eventually, the two of you will be fighting for control of the entire territory. Until then, the collection of taxes is crucial to your military build-up. You must make sure that you collect more tax money than your rival, and that you outstrip him by as much as possible.

Your advisor has studied the records kept by the scribes of the former King of this territory, and so has been able to estimate the tax revenue that can be expected from each square. Based on this, you want to select the site for your own castle that guarantees the best possible advantage taxes no matter what space your rival baron may select.

NOTE: Distances are Euclidean (distance between the center of two squares as the crow flies).

Input

The first line of input will contain two integers, w, and h, denoting the width and height (in numbers of squares) of the map. w and h will be in the range 1–50, inclusive.

This is followed by $w \times h$ integers distributed across an arbitrary number of subsequent lines. Each of these represents the expected tax collection (in gold pieces per year) for one map square. They occur in the order:

$$(0,0)(1,0),\ldots,(w-1,0)(0,1)(1,1),\ldots,(w-1,h-1)$$

Each item will be in the range 0–40, inclusive. A value of 0 denotes water or land that is otherwise uninhabitable—castles cannot be built on those squares.

All maps used as input in this problem will be large enough to guarantee that both castles can be placed on a non-zero square, no matter where the first one is placed (i.e., you cannot crowd your rival entirely off the map).

Output

Output from this program consists of a single line of the form:

```
Place your castle at: X Y
```

where X and Y are two integers, separated by a single space, denoting the optimal placement of your castle, indexed from 0.

If there is more than one location on the map that may be chosen to achieve the same maximal advantage over your rival, any one of those positions will be an acceptable answer.

Example

Given the input

```
7 7
3 4 1 0 0 0 0
2 1 1 0 0 0 0
1 1 1 1 0 0 0
1 1 1 0 1 1 1
0 0 0 1 1 1 2
0 0 0 0 1 2 1
0 0 0 0 1 3 4
```

Your program should print:

```
Place your castle at: 3 4
```

Problem F: In Defence of a Garden

It's said that "Necessity is the mother of invention," but some people think that "Laziness" is a more likely parent.

Hubert Greenthumb hated digging fence posts. But he knew that, without a fence around his garden, deer from the nearby woods would eat his vegetables before he could harvest them.

Being something of a tinkerer, he retired to his workshop with a small garden tractor, some out-of-date computer chips, and a couple of robot arms he had picked up at a bankruptcy auction from a failed ".com" high-tech company. After two days of work, he emerged as the proud inventor of the Greenthumb Automatic Garden Fence Layer (pat. pending).

To his skeptical wife (who observed that he could easily have built the fence in half the time it took to construct this machine), he explained that he needed only to program in the desired fence shape, and the machine would proceed to chug around the yard, laying down a fence in 1-foot sections until the job had been completed.

Hubert proceeded to key in instructions to enclose a square area, 25 feet on a side, of his 100' by 100' yard. He set the machine to operating and went inside for a celebratory drink.

When he emerged, he discovered that the machine had laid down fence in an elaborate, possibly random walk about his lawn. Unwilling to actually admit that anything had gone wrong, he announced his intention to plant within the garden actually laid out by the machine, as if he had wanted it that way all along. Any section of the yard that was no longer accessible to the deer (enclosed by the fence) would be considered as garden space.

"Fine," sighed his wife, "but we'll need to know just how many square feet of garden we have so that we can buy an appropriate amount of seeds." Hubert gamely began to trace out the fence laid down by the machine. "Let's see, it went North for 5 feet, then West for 3 feet, ..."

Note: Hubert's yard can be divided into a grid of 100×100 feet, with each grid box being 1 foot by 1 foot. The robot moves along the edges of the boxes. As the robot moves, it builds a fence from vertex to vertex of the grid (intersections of the lines).

Note: Because the robot moves along the edges of the grid, you can ignore the amount of space the fence occupies. For example, if the robot moves North one, East one, South one, and West one, it has enclosed one square foot of garden space.

Input

The first line of the input will contain the number of data sets. The are no blank lines before or after each data set.

The first line of each data set will contain three integers (X Y Z), indicating the X and Y position of the starting point on the grid, and the number of moves the robot makes. (X, Y, and Z are all non-negative integers, X is the number of feet from the western edge, Y is the number of feet from the southern edge of the yard). (X and Y range from 0 to 100, inclusive.)

The next Z lines will contain a character D and an integer F, separated by a space. The character will indicate the direction (`N`, `S`, `E`, `W`) and the integer will indicate how far in that direction the robot traveled.

The path will never leave the 100' by 100' yard. The path may or may not be closed. It may cross itself or retrace its steps (walk along lines in the grid it previously laid fence). It automatically stops building fence until it moves onto an edge without fence on it.

Output

For each data set, output a single line of the form

```
Data Set N: Q square feet.
```

where N is the data set number (from 1) and Q is the number of square feet that are enclosed so they may be used for the garden.

After the last line of output, print "End of Output" on a line by itself.

Example

For the input:

```
1
0 0 8
N 25
E 25
N 25
E 25
S 25
W 25
S 25
W 25
```

The output is:

```
Data Set 1: 1250 square feet.
End of Output
```

Problem G: Cantoring Along

The Cantor set was discovered by Georg Cantor. It is one of the simpler fractals. It is the result of an infinite process, so for this program, printing an approximation of the whole set is enough.

The following steps describe one way of obtaining the desired output for a given order Cantor set:

1. Start with a string of dashes, with length 3^{order}
2. Replace the middle third of the line of dashes with spaces. You are left with two lines of dashes at each end of the original string.
3. Replace the middle third of each line of dashes with spaces. Repeat until the lines consist of a single dash.

For example, if the order of approximation is 3, start with a string of 27 dashes:

```
---------------------------
```

Remove the middle third of the string:

```
---------         ---------
```

and remove the middle third of each piece:

```
---   ---         ---   ---
```

and again:

```
- -   - -         - -   - -
```

The process stops here, when the groups of dashes are all of length 1. You should not print the intermediate steps in your program. Only the final result, given by the last line above, should be displayed.

Input

Each line of input will be a single number between 0 and 12, inclusive, indicating the order of the approximation. The input stops when end-of-file is reached.

Output

You must output the approximation of the Cantor set, followed by a newline. There is no whitespace before or after your Cantor set approximation. The only characters that should appear on your line are '-' and ' '. Each set is followed by a newline, but there should be no extra newlines in your output.

Example

For the input:

```
0
1
3
2
```

The output is:

```
-
- -
- -   - -       - -   - -
- -   - -
```

Problem H: Chambers Ceramic Conundrum

The Chambers Construction Company has a major contract to deliver a tile floor on schedule for its largest customer (Moneybags to Spare Inc). Unfortunately, the clerk who ordered tiles won the lottery just after ordering the tiles for this room, and did not write down where to place each tile to make the room fit. Normally this would not be a problem, except the tiles that were ordered are not all squares. They are each made up of 4 square segments, but will take on all possible shapes shown here:

```
XXXX     XX       XX        XX      X       XXX       X
         XX        XX      XX      XXX      X       XXX
```

Given that the project is under an extremely tight schedule, it is not possible to reorder the tiles in a more standard manner. Instead when the 9 tiles come in, you will need to figure out how to place the tiles (or if there is no way to set the pieces correctly). The tiles in the box are ordered from A to I. The room that needs to be tiled is 6 segments on each side.

Placing Tiles

A middle-manager at CCC has come up with an algorithm that they will give to the tiler to tile the room. The tiler has come to you to write a program to determine what pattern will be successful without having to try all of them with the physical tiles. The tiler will always start with the top left corner of the room. After placing the first tile, they will work their way from left to right and from top to bottom, placing the next tile such that it will fill the leftmost open space on the top line with an open space.

For instance, if the layout currently had the following tiles (A and B) placed:

```
AABBBZ
 AA  B
```

The next tile to place would be placed such that position Z is filled.

To make matters easiest, the tiler will always place the earliest tile in the box that could successfully fill that position. For instance, if either tile B or C could fill that position, the tiler will choose B. Furthermore, they will always place the tile as they are oriented above if possible. They will then attempt to rotate the tile 90 degrees clockwise and place it (possibly doing this 3 times).

NOTE: Remember the tiles may be rotated, but they may not be flipped.

A tile may not extend outside the 6x6 room, or overlap with another tile. A room is considered successfully tiled if the given set of tiles completely tile the room using the above algorithm.

The tiler will continue laying tiles according to this algorithm until finishing the room, or discovering that the room cannot be finished using the previous choices. If the room cannot be finished, the tiler will backtrack, considering the remaining rotations of the previous tile, and then the remaining tile. The tiler will continue to backtrack, one tile at a time, until all combinations have been tried or the room cannot be successfully tiled.

Input

The first line of input will indicate how many data sets are included. (N)

The next N lines will each contain 9 numbers. These numbers indicate the shape of each tile The first entry on the line will correspond to tile A, the second to tile B, ..., the 9th for tile I. Each number will reflect the layout of a tile as specified above (the left most layout is 1, the rightmost is 7).

Output

For each data set, the first line of output should indicate the index of the data set, starting with 1. ("Data Set 1")

The next line of input will indicate the floor may or may not be tiled successfully. ("The floor may be tiled." or "The floor may not be tiled.").

The next 6 lines would then display the graph of floor illustrating the final layout. To make it easy to understand, each tile set should be marked A–I corresponding to the order they were in the input line. Each line will have exactly 6 characters indicating the tile segment, followed by a newline.

A blank line should appear after each data set.

The line stating "End of Output" should appear after the last data set.

Example

Input:

```
2
2 2 2 2 2 2 2 2 3
1 1 2 1 1 2 1 1 2
```

Output:

```
Data Set 1
The floor may not be tiled.

Data Set 2
The floor may be tiled.
AAAABD
CCEGBD
CCEGBD
FFEGBD
FFEGII
HHHHII

End of Output
```