1. Bank Account

Program Name: Bank.java Input File: bank.dat

Scrooge McDuck wants to move into the 21st century and wants a program to help him keep track of his gold in his new account. He wants the program to simulate what the bank is doing, interest and all, to make sure they don't stiff him a single penny! His account pays 2% interest on the first of every month based on the balance at the end of the previous month and is initially empty. The bank also truncates, not rounds, overall balances for output, but keeps full precision internally at all times. Every time he makes a deposit or withdrawal he will register the amount, preceded by the date, into an account file known as the transaction log. Your program should read his transaction log and print out the balance his account should have.

Input

The file will contain an unknown number of lines. On each line there will be a date in YYYY-MM-DD format, a space, and then a positive or negative amount to hundredths precision indicating a deposit or withdrawal, respectively. The entries are in chronological order as shown below.

Output

You should print a dollar sign followed by the balance in his account as of the last date with activity in the transaction log truncated to the hundredths digit, as shown below.

Example Input File

```
2010-01-11 100.00
2010-01-30 100.00
2010-02-01 100324.32
2010-03-13 13243.53
2010-03-14 -1343.43
2010-06-11 -32.43
2010-11-11 1000.00
2010-12-31 100000.00
2011-01-30 1.00
2011-02-28 -13.33
```

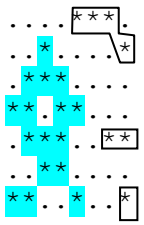
Example Output to Screen

```
$247341.39
```

2. Blob Size

Program Name: BlobSize.java Input File: blobsize.dat

Jeffery is studying different shapes in a plane. For this particular study, he refers to the shapes as "blobs" because they are irregularly shaped polygons that remind him of blobs even though some of the blobs have centers that are not part of the blob. He represents his blobs in a rectangular grid as a collection of one or more contiguous asterisks (`*`). Contiguous means that the asterisks must be adjacent either horizontally, vertically or diagonally. Characters in the grid that are not part of a blob are represented by periods (`.`). In the diagram below, there are 4 blobs – three shown with boxes and one shown grayed.



You are to write a program that will determine the number of cells in the blob given the coordinates of a particular character in the grid. The uppermost, leftmost character of the example above is in row 1, column 1 or `1 1`.

Input

The first line of input will contain a single integer `n` that indicates the number of data sets to follow. For each data set:

- the first line will contain three integers in the form `r c s` which meet the following criteria:
  - `r` ≥ 3 is the number of rows in the grid
  - `c` ≥ 3 is the number of columns in the grid
  - `s` > 1 is the number of test cases for that grid
- the next `r` lines will contain the grid.
- the next `s` lines will each contain an ordered pair `x y`, 1 ≤ `x` ≤ `r` and 1 ≤ `y` ≤ `c`, which is the location of a character in the grid.

Output

For each test case, you will print the number of cells in the blob that contains the cell at the given location. If the test case falls on a square that is not part of a blob, print `NO BLOB`.

Example Input File

```
2
7 8 2
....***.
..*....*
.***....
**.**...
.***..**
..**....
**..*..*
7 1
5 6
4 8 3
......**
***.***.
......**
***.***.
2 3
4 5
2 7
```

2. Blob Size (cont.)

Example Output to Screen

```
16
NO BLOB
3
10
10
```

3. DVD Elevator Algorithm

Program Name: DVDelevator.java Input File: dvdelevator.dat

You have taken an internship at Flaming Monkey Games working on a launch title for the new Phony PlayBox video game console. The game is currently loading too slowly and the lead programmer has asked you to implement an elevator loading algorithm to speed it up.

Data on DVDs are stored in a linear fashion from the inside to the outside of the disk. The laser can move either way across the disk, but always reads data in multiples of sector sizes as it moves from inside to outside. It takes time to move the laser, and the size of the sectors and the amount read per second depends on the speed of the drive. You don't have direct control over the laser; you can only send file read requests to the operating system. You can also assume the operating system starts by having the laser at the beginning of the first sector.

The PlayBox has an optical disk with the following specifications:

- disk size:
  - there are 131,072 sectors with a sector size of 64 kilobytes
  - total disk size is 8 gigabytes of data, where
    - 1024 bytes = 1 kilobyte
    - 1024 kilobytes = 1,048,576 bytes = 1 megabyte
    - 1024 megabytes = 1,073,741,824 bytes = 1 gigabyte
- disk speed:
  - the laser takes 1 microsecond to swipe over a sector in either direction, whether it is reading data or not.
  - overall drive speed is 20 megabytes per second, where
    - 1000 microseconds = 1 millisecond
    - 1000 milliseconds = 1 second

An elevator algorithm is used to order requests based on the motion of the laser in order to minimize laser motion and increase disk read throughput. Your boss has given you explicit instructions on how to implement the algorithm. The instructions are:

1. By default the algorithm should be in an "idle" state while waiting for read requests.
2. When a request is received, it should wait 10 milliseconds and lump all other requests it receives during that time into a single request batch. Requests received at the exact end time of the batch time window should be included inside the batch.
3. After the time expires it should dispatch all the requests to the operating system in sector order so all the requested data can be read as fast as possible without additional seeks. It should also keep track of the sector the laser will stop on as an optimization for new reads.
4. The system should go back to "idle" until more reads come in.
5. When a new read comes in it should be checked against the current laser sector. If the new read is at or past the current laser sector the request should be dispatched right away. If not then the algorithm should go back to #2.

The lead has asked that you test your code by writing a program that will emulate what will happen in the game by reading in read requests from a file and running them through your algorithm. The program will also have to emulate what a DVD drive will do in order to be accurate. Optimal output will also be provided so that you can make sure your algorithm is as efficient as possible.

Input

The input file will contain an unknown number of lines. Each line will contain 3 numeric values separated by a single space:

- the time in microseconds since the start of the game until the request was made, in ascending order
- the start sector for the request, and
- the end sector for the request.

3. DVD Elevator Algorithm (cont.)

Output

You should output all the requests from the input file, one per line, in the order they are processed by the DVD hardware. Each line should be of the form "`Time X: Y to Z`" where `X` is the time in microseconds since the start of the game until the DVD hardware processes the request, and `Y` and `Z` are the start and end sectors respectively.

Example Input File

```
10 2 200
100 300 400
900 210 220
1100 240 270
10000 220 240
10002 10000 20000
11003 30000 50000
20000 80000 90000
20002 20000 30000
```

Example Output to Screen

```
Time 10011: 2 to 200
Time 10219: 210 to 220
Time 10229: 220 to 240
Time 10249: 240 to 270
Time 10309: 300 to 400
Time 20009: 10000 to 20000
Time 40009: 30000 to 50000
Time 90009: 80000 to 90000
Time 180009: 20000 to 30000
```

4. EXCELlent

Program Name: Excellent.java Input File: excellent.dat

Ms. Jones spends a lot of time with spreadsheets. One of her tasks is to compare two Excel spreadsheets to find the names in spreadsheet 1 that are missing from spreadsheet 2 and the names in spreadsheet 2 that are missing from spreadsheet 1.

You are to write a program that examines the spreadsheets and prints lists of the missing names.

Input

The first line of input will contain two integers, `p` and `q`, where `p` indicates the number of names in spreadsheet 1 and `q` indicates the number of names in spreadsheet 2. The next `p` lines will contain the names in spreadsheet 1 followed by `q` lines containing the names in spreadsheet 2. The names will be all uppercase in the form: LAST NAME, FIRST NAME.

Output

You will print the heading `NAMES IN SPREADSHEET 1 THAT ARE NOT IN SPREADSHEET 2` followed by the list of names in spreadsheet 1 that are not in spreadsheet 2 in alphabetical order, one per line. After this list, you will print the heading `NAMES IN SPREADSHEET 2 THAT ARE NOT IN SPREADSHEET 1` followed by the list of names in spreadsheet 2 that are not in spreadsheet 1 in alphabetical order, one per line.

Example Input File

```
10 9
SMITH, MARY
JONES, KERRY
ROBERTS, AMBER
JONES, JERRY
LASITER, RICK
ROGERS, GENE
ANDERSON, TROY
HARWOOD, SHIRLEY
GARCIA, RICARDO
GREEN, WES
SMITH, MARY
DOLE, ANN
JONES, KERRY
ROBERTS, AMBER
SMALLWOOD, LESTER
JONES, JERRY
ROGERS, GENE
RODRIGUEZ, JORGE
GARCIA, RICARDO
```

Example Output to Screen

```
NAMES IN SPREADSHEET 1 THAT ARE NOT IN SPREADSHEET 2
ANDERSON, TROY
GREEN, WES
HARWOOD, SHIRLEY
LASITER, RICK
NAMES IN SPREADSHEET 2 THAT ARE NOT IN SPREADSHEET 1
DOLE, ANN
RODRIGUEZ, JORGE
SMALLWOOD, LESTER
```

5. Family Tree

Program Name: Family.java Input File: family.dat

You feel compelled to write a program that can read in a text file containing a list of relationships and parse that data into a family tree. For now you are happy with a first pass that can print out generational information to a console. Later on, however, you will want to expand on this and create a graphical representation for the tree.

Input

The input file will contain an unknown number of lines. Each line will contain 3 words separated by a space: the source name, the description noun, and the destination name. The description noun describes how the source is related to the destination. The nouns can be:

- MOTHER
- FATHER
- DAUGHTER
- SON
- SISTER
- BROTHER
- WIFE
- HUSBAND

Everybody in the tree will have a unique name, and the family tree is guaranteed to be legal in all 50 states. It is possible for the same person to show up multiple times in the input file, but the same relationship between two people will only show up once. You can also assume that any two people in the tree have a path that joins them and marriage occurs on the same generation.

Output

The output is a generation per line that contains the generation count and the names of every member of that generation in alphabetical order, as shown below.

Example Input File

```
Bob SON Mary
Josephus HUSBAND Mary
Brutus FATHER Mary
Vicky MOTHER Mary
Josephus SON Gary
Matilda MOTHER Josephus
```

Example Output to Screen

```
1st Generation: Brutus Gary Matilda Vicky
2nd Generation: Josephus Mary
3rd Generation: Bob
```

6. Frogger

Program Name: Frogger.java Input File: frogger.dat

Wes is writing a computer application that involves a tadpole getting out of a maze of walls. The tadpole is given an initial starting position in a rectangular maze, which for the purpose of this program is depicted as an `r` x `c` matrix of cells. The tadpole can swim from one cell to a horizontally or vertically adjacent cell, avoiding the walls, until he gets out of the maze by landing on the exit cell.

However, if the tadpole moves into a "magic" cell (a cell denoted by a single digit *d*), he temporarily turns into a frog. A frog has special frog powers that enable him to jump either horizontally or vertically over *d* number of cells and/or walls. If, after he jumps, the frog

- lands on the exit cell, he gets out of the maze.
- lands on an empty cell, he turns back into a tadpole and continues.
- lands on another magic cell, he remains a frog and can jump to another cell if possible.

If the frog cannot jump to one of the above three kinds of cells, he returns to his tadpole state with no magic powers and can swim to another cell if possible.

The starting cell is considered to be the first cell visited by the tadpole. A cell is considered visited when either the tadpole or frog enters it. Neither the tadpole nor the frog

- can visit or jump to a previously visited cell.
- can visit or jump to a wall.
- can go outside the maze.

Input

The first line of input will contain a single integer `n` that indicates the number of mazes to follow. For each maze, the first line will contain the size of the maze in the form `r c` where $3 \le r \le 8$ is the number of rows and $3 \le c \le 8$ is the number of columns in the maze. The next `r` lines will contain `c` characters as follow:

'*****' is a wall

'**.**' is an empty cell to which the tadpole or frog can move.

`d` is a digit in the range 1 through 4 which indicates that the tadpole has turned into a frog. `d` is the number of cells (including wall cells) that the frog can jump over, if possible, before turning back into a tadpole. For example, if `d` = 3, the frog would land on the 4th cell from where he started.

'$' is the tadpole's starting position

'@' is the cell that the tadpole will use to exit the maze

Output

For each maze, you will print the minimum number of cells in the maze that the tadpole will visit before exiting the maze.

Note: Some mazes could take an extraordinarily long time to solve. The data sets for this problem have been designed to execute in less than 5 seconds. Solutions taking longer than 30 seconds to execute will be rejected.

6. Frogger (cont.)

Example Input File

```
2
8 8
******..
....$...
..***.2.
.3..***.
**.2.***
***.....
....@...
****..**
5 7
****$**
**....2
.*..3..
.*....*
@..****
```

Example Output to Screen

```
8
6
```

7. Insertion Sort

Program Name: Insertion.java Input File: insertion.dat

Rick wants you to write a program for him that will sort the playing cards that he has in his hand in order of increasing value: 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace. His hand might have more than one card with the same value from different suits, but the card suit does not matter to him. You are to write a program that uses the standard insertion sort to sort the cards for him just by value.

The algorithm for a standard insertion sort is:

1. Consider the first card in the hand to be a sorted array of length one and the remaining cards to be an unsorted array of length n-1, where n is the number of cards to be sorted.
2. Remove the first card from the unsorted array and place it into the sorted array in the position that will maintain the sorted array's order and move the sorted cards by one position as needed. This increases the sorted array's length by one and decreases the unsorted array's length by one.
3. Repeat step 2 until all cards are contained in the sorted array and the unsorted array is empty.

Rick wants to trace what is happening with his data when he uses this insertion sort to sort a non-empty array of cards. You are to write a program for him that will print the order of his cards after each iteration of the insertion sort.

Input

The first line of input will contain a single integer `n` that indicates the number of hands of cards to follow. Each of the following `n` lines will contain the cards contained in a single hand. The cards will be denoted by the characters `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `T`, `J`, `Q`, `K`, `A` which represent 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King and Ace respectively. The characters will all be separated by a single space.

Note: You may assume that each hand will contain at least 2 cards.

Output

For each array, you will print the state of the hand of cards after each iteration of the standard insertion sort. Print a blank line after the last iteration through each hand.

Example Input File

```
2
8 3 5 7 2 A 4
5 3 7 2 9 T A 2 5
```

Example Output to Screen

```
3 8 5 7 2 A 4
3 5 8 7 2 A 4
3 5 7 8 2 A 4
2 3 5 7 8 A 4
2 3 5 7 8 A 4
2 3 4 5 7 8 A

3 5 7 2 9 T A 2 5
3 5 7 2 9 T A 2 5
2 3 5 7 9 T A 2 5
2 3 5 7 9 T A 2 5
2 3 5 7 9 T A 2 5
2 3 5 7 9 T A 2 5
2 2 3 5 7 9 T A 5
2 2 3 5 5 7 9 T A
```

8. MipMapping a Texture

Program Name: Mipmapping.java Input File: mipmapping.dat

In computer graphics images are stored as textures, which are two dimensional arrays of texel data. A texel is just an individual pixel of a texture. Each texture will have its own height and width, and can have different formats (how the texel data is actually stored). There are many different texture formats which all have a different data size to visual fidelity ratio in order to serve different purposes, such as computer games or web page images. We will be focusing on standard, uncompressed data for simplicity. The format will be described below.

Mipmaps are smaller versions of a texture that are used by graphics hardware to speed up rendering at a distance. As the texture moves farther away the hardware chooses a smaller mip level, if available, in order to save room in the texture cache and decrease texture bandwidth usage. Since the image is taking up a smaller amount of screen space, the smaller texture fits better with less scaling and one cannot notice the decrease in quality. These mipmaps are stored along with the main texture in a special texture format. The hardware requires that each of the textures' dimensions be of powers of 2 if they are to have mipmaps, and each mipmap has dimensions of powers of 2 down from the previous. The mipmap chain stops when one dimension hits 1. For example, a 16 by 16 texture will have an 8 by 8 mipmap, a 4 by 4 mipmap, a 2 by 2 mipmap, and a 1 by 1 mipmap. A texture whose width or height is not a power of 2 cannot be mipmapped.

These textures are stored as 32 bits (4 bytes) per pixel, or texel. Each set of 4 bytes in a texel represents a color channel. In order the texels are *Alph*a (how transparent the texel is, with 0 being transparent and 255 being fully opaque), *Red* (0 being no red contribution, 255 being full red), *Green* and *Blue*. Opaque purple would be represented as `0xffff00ff` and transparent white would be `0x00ffffff` (all colors set to full creates white on computers).

Your boss at work needs you to write a program that can generate mipmaps for a 2D texture based on your new knowledge. For each texel of the mipmap you are generating, you should take a 2 by 2 block of texels from the mipmap above it, average each one of the color channels individually, and then combine those bytes. For example, the two colors `0x88222222` and `0x22444444` would make `0x55333333`. You shouldn't worry about manually rounding but instead let the Java integer and bit shifting math do it for you by dropping bits.

Input

The first line of input will contain a single integer `n` that indicates the number of data sets to follow. For each data set:

- the first line will contain two integers `h` and `w` that indicate the height and width of a texture
- the next `h` lines will contain the texture
- each texture line contains `w` integers separated by a single space
    - each integer will be a hexadecimal 32 bit integer with 32 bit precision
    - each integer will be in the format `0xaarrggbb`, where `a` is Alpha, `r` is Red, `g` is Green and `b` is Blue.

Output

For each data set containing a texture with dimensions that are powers of 2, you will print all mipmaps, with one row of texels per line. The texels should be printed out separated by a space in full 32 bit precision and using lowercase alpha characters, just like the input. There should not be any spaces between the mipmaps but there should be a blank line after each texture, as shown below. For a data set containing a texture that cannot be mipmapped print `NO REDUCTION POSSIBLE` followed by a blank line.

8. MipMapping a Texture (cont.)

Example Input File

```
2
4 4
0xffffffff 0xffffffff 0x00000000 0x00000000
0xffffffff 0xffffffff 0x00000000 0x00000000
0x00000000 0x00000000 0xffffffff 0xffffffff
0x00000000 0x00000000 0xffffffff 0xffffffff
8 8
0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc
0x7fcccccc 0xffff0000 0xffff0000 0xffff0000 0xffff0000 0xffff0000 0xffff0000 0x7fcccccc
0x7fcccccc 0xffff0000 0x00000000 0xffffff00 0x00000000 0xffff0000 0xffff0000 0x7fcccccc
0x7fcccccc 0xffffffff 0x00000000 0xff0000ff 0x00000000 0xffffffff 0xffffffff 0x7fcccccc
0x7fcccccc 0xffffffff 0x00000000 0xff0000ff 0x00000000 0xffffffff 0xffffffff 0x7fcccccc
0x7fcccccc 0xff00ff00 0x00000000 0xff00ffff 0x00000000 0xff00ff00 0xff00ff00 0x7fcccccc
0x7fcccccc 0xff00ff00 0xff00ff00 0xff00ff00 0xff00ff00 0xff00ff00 0xff00ff00 0x7fcccccc
0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc 0x7fcccccc
```

Example Output to Screen

```
0xffffffff 0x00000000
0x00000000 0xffffffff
0x7f7f7f7f

0x9fd89999 0xbfe56666 0xbfe56666 0x9fd89999
0xbfe5a5a5 0x7f3f3f3f 0x7f3f3f3f 0xbfe5a5a5
0xbfa5e5a5 0x7f003f7f 0x7f003f7f 0xbfa5e5a5
0x9f99d899 0xbf66e566 0xbf66e566 0x9f99d899
0xa7b87878 0xa7b87878
0xa769b888 0xa769b888
0xa7909880
```

9. Music CD

Program Name: Music.java Input File: music.dat

Alexandra is making some music CDs to use at a party. Each CD will contain no more than 20 minutes worth of songs. She has made a list of songs that she wants to use and the order in which she wants to play them. You are to write a program that will tell her which songs will be on each CD and the total playing time of the CD.

Input

The first line of input will contain a single integer `n` that indicates the number of songs she has on her list. Each of the following `n` lines will contain the name of a song followed by two integers, `m` and `s`, that indicate the length of the song in minutes `m` and seconds `s`. Each of items will be separated by a space.

Output

For each CD, you will output `CD #x`, where `x` is the number of the CD, followed by the first and last song on the CD and the total number of minutes and seconds of songs on the CD in the format `mm:ss` for times 10 minutes or greater and `m:ss` for times less than 10 minutes. All information shall be printed on separate lines with a blank line after the information for each CD.

Example Input File

```
21
Can't You Hear Me Knocking 3 45
Sunshine of Your Love 4 12
While My Guitar Gently Weeps 4 23
Riders on the Storm 4 12
Back in Black 5 12
Smoke on the Water 3 56
Dance the Night Away 3 23
The Best Is Yet to Come 3 46
The Way You Look Tonight 3 32
Luck Be a Lady 4 21
Bewitched 3 42
The Good Life 3 56
The Girl from Ipanema 3 43
Fly Me to the Moon 2 34
Summer Wind 3 32
Strangers in the Night 3 52
Call Me Irresponsible 4 3
Somethin' Stupid 5 2
My Kind of Town 5 42
It Was a Very Good Year 5 21
That's Life 13 21
```

9. Music CD (cont.)

Example Output to Screen

```
CD #1
Can't You Hear Me Knocking
Riders on the Storm
16:32

CD #2
Back in Black
The Way You Look Tonight
19:49

CD #3
Luck Be a Lady
Fly Me to the Moon
18:16

CD #4
Summer Wind
Somethin' Stupid
16:29

CD #5
My Kind of Town
It Was a Very Good Year
11:03

CD #6
That's Life
That's Life
13:21
```

10. Ones Galore

Program Name: Ones.java Input File: ones.dat

Any positive integer that is not divisible by 2 or 5 has a multiple that is all ones. For example, for the integer three, 111 is a multiple that contains all ones. You are to write a program that will find the smallest integer containing all ones that is a multiple of a given integer.

Input
The only line of input will contain a list of positive integers less than 20, each separated by a space.

Output
For each integer input, you will print on a separate line the number of ones in the smallest multiple of the integer that contains only ones. If the input is not valid, print `INVALID INPUT`.

Example Input File
```
7 5 9 17
```

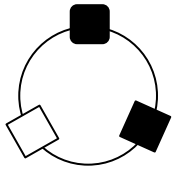
Example Output to Screen
```
6
INVALID INPUT
9
16
```

11. Rings

Program Name: Rings.java Input File: rings.dat

Archaeologists have uncovered some ancient rings containing black and white stones near an ancient campsite in east Texas. It is believed that these rings were used as a method to record a person's "worth" using a binary number system. However, it is not known if white stones represent a one and the black stones represent a zero or if the black stones represent a one and the white stones represent a zero. It is also not known where the starting point is located on the ring.

You are to write a program that will determine the minimum and maximum "worth" of each ring in base 10. For example, the ring could contain these rocks:



Depending on the starting point, the rings could be read as follows with these values where `B` is for black and `W` is for white:

Order if read clockwise	**Worth if `B` is 1 and `W` is 0**	**Worth if `W` is 1 and `B` is 0**
BBW	6	1
BWB	5	2
WBB	3	4
Order if read counter-clockwise		
BWB	5	2
WBB	3	4
BBW	6	1

In this case, the maximum worth is 6 and the minimum worth is 1.

Input

The first line of input will contain a single integer `n` that indicates the number of rings to follow. Each of the following `n` lines will contain a string composed of at least one but no more than thirty `W` and `B` characters indicating white rocks and black rocks, respectively. Each string represents a ring whose starting point and reading order is unknown, as described above.

Output

You will output the minimum value, a space, and the maximum value for each ring.

Example Input File

```
4
BBW
BBBW
BBWBB
BWWBB
```

Example Output to Screen

```
1 6
1 14
1 30
3 28
```

12. Roll

Program Name: Roll.java Input File: roll.dat

Kim and Pat like to text each other but are afraid that their parents might read their text messages. They have decided to write in code so their parents would not know what they are saying. They have developed a method for encoding their messages. The process for their code is:

- Put the message, including spaces, in the smallest square matrix that will hold the message. Fill cells in each row from left to right before moving to the next row. Rows and columns are numbered beginning with row one and column one as shown below.
- If there are unused cells at the end of the matrix, the first cell will be filled with an asterisk (*) and the remaining unused cells will be filled with consecutive letters of the alphabet beginning with the letter A and continuing until all empty cells have been filled.
- "Roll" the matrix to encode the message as follows:
  - cells in a square with its left, uppermost cell having an odd numbered row and column (e.g. 1,1 or 3,3) will rotate each letter one cell clockwise around the square.
  - cells in a square with its left, uppermost cell having an even numbered row and column (e.g. 2,2 or 4,4) will rotate each letter one cell counter-clockwise around the square.
- Rewrite the coded message in the new order by rows and send the coded message.

For example, the message **I love Computer Science** would fit into the 5x5 matrix shown at the left below and after the "Roll" the matrix would look like the matrix shown on the right below. All the cells in the square beginning with 1,1 have been rotated one position clockwise. All the cells in the square beginning with 2,2 have been rotated one cell counter-clockwise, etc.

	1	2	3	4	5
1	I		L	O	V
2	E		C	O	M
3	P	U	T	E	R
4		S	C	I	E
5	N	C	E	*	A

	1	2	3	4	5
1	E	I		L	O
2	P	C	O	E	V
3			T	I	M
4	N	U	S	C	R
5	C	E	*	A	E

The encoded message sent would be: `EI LOPCOEV  TIMNUSCRCE*AE`

You are to write a program that will decode the encoded message.

Input

The first line of input will contain a single integer `n` that indicates the number of encoded messages to be sent. Each of the following `n` lines will contain a single encoded message with a perfect square number of characters less than 170 characters long.

Output

For each encoded message, you will print the decoded message. Do not include the characters used to fill the matrix.

Example Input File

```
2
EI LOPCOEV  TIMNUSCRCE*AE
TSOLVERIS E  HOBWPNO LLMST UIITATE*
```

Note: There is a space after the `*` in case #2

Example Output to Screen

```
I LOVE COMPUTER SCIENCE
SOLVE THIS PROBLEM TO WIN UIL STATE
```