University of Manitoba
Open Programming Contest
September 26, 2009

General Instructions:

1. Submit solutions using the PC^2 software, as demonstrated.
2. The questions are not listed in order of difficulty. Some questions will be easier than others, and these are not necessarily the problems listed first.
3. All input should be read from standard input.

Problem 1 – Little League Light Cycles

In this game, you will play a modified version of light cycles (from the 1982 science fiction movie Tron). In the game, you and an opponent are each assigned a starting and finishing point. The goal is to reach your finishing point safely. You only travel in a single straight line, and you cannot cross a path previously taken by your opponent. You and your opponent both travel at the same speed.

There are three possible outcomes of the game:

- you win, by reaching your finishing point while your opponent cannot, since doing so would require them to cross your path.
- your opponent wins, for the analogous reason.
- you can both reach your end points without crossing any paths. This is a draw.

You may assume that the case where both you and your opponent crash into each other at the same time does not occur, nor do you and your opponent travel along paths which are part of the same line.

Given several rounds of light cycles, you should determine for each if you win, your opponent wins, or there is a draw.

Input

The first line gives `n`, the number of tests in the input. Each of the next `n` lines represents one test. The lines contain eight floating point numbers: `x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4,` each separated by a single space. The pair `(x_1,y_1)` represents your starting point, `(x_2,y_2)` is your finishing point, `(x_3,y_3)` is your opponent's starting point and `(x_4,y_4)` is your opponents finishing point.

Output

For each test, write either `"I win"`, `"I lose"` or `"Draw"`.

Sample Input	Sample Output
3 1.0 1.0 3.0 3.0 4.0 0.0 1.0 3.0 3.0 0.0 2.0 4.0 1.0 1.0 3.0 4.0 1.0 3.0 0.0 1.0 1.0 1.0 3.0 2.0	I win I lose Draw

Problem 2 – Scientific Notation

In Java, the expression

```
double x = 3.14159e10;
```

is a valid expression, and denotes that `x` has the value `31415900000`. Write a program that converts expressions of the form `xey`, where `x` is a floating point number and `y` is an integer (positive, not zero) into an equivalent floating point number without the `e`. The value should be `x * 10`^{y}.

Input

The first line gives `n`, the number of tests in the input. The next `n` lines consist of scientific notation expressions to be converted. They consist of a single digit (`0-9`), followed by a decimal point, and then followed by at least one and at most 8 digits (`0-9`). The next character is an e, followed by at least one and at most 4 digits. The last four digits are not all zero and do not contain leading zeroes.

Output

For each scientific notation expression, output the equivalent number without scientific notation. The numbers should be properly formatted

Sample Input	Sample Output
3 3.14e2 3.14e3 3.14e15	314 3140 3140000000000000

Problem 3 – Buying Candles

Mike is a tremendously cheap. In an effort to (he believes) buy as few birthday candles for his son as possible, he buys candles shaped like `0`s and `1`s, and displays his son's age in binary. So, on his son's fifth birthday, the candles read "`101`".

Mike wants to know how many candles he will need to buy each year. He has determined that each candle will last exactly two years. Mike calls the value for year `n` the *`n`-th candle* number. For example, on his son's first birthday, he grudgingly bought a `1` candle. The next year, his `1` candle was still good, but now he needed a `0` candle! So the first two values of the candle sequence are 1 and 1.

Input

The first value `m` is the number of test cases. The next `m` lines each consist of a single integer `n`.

Output

For each integer `n`, give the `n`-th candle number.

Sample Input	Sample Output
5 1 2 3 4 5	1 1 2 1 1

Problem 4 – Omicronian Refueling

You are a frequent commuter along a long road, which requires you to fuel up several times. Unfortunately for you, the gas stations are all owned by a technologically advanced but impatient species, the Omicronians. The Omincronians have automated pumps, but when you want to fill your tank, you are charged for an entire tank, regardless of how much fuel you actually need, and all the fuel is pumped into your car (potentially causing an overflow). **Fuel costs one dollar per litre.**

And to make matters worse, the Omicronians charge a penalty for overflows: for x litres of spilled fuel, the penalty is x^2 dollars.

You have calculated the amount of fuel it takes to drive from one station to the next. This never changes. You always begin your trip with a full tank, and you can always travel from one station to the next with a full tank of gas. You would like to the minimum possible cost of driving from one end of the road to the other.

Input

The first line gives n, the number of tests in the input. Each of the next n pairs of lines represents one test. The first of the pair lines contain an integer Q (the capacity of the tank – also the cost of a full tank of fuel, not including the penalties) and r, the number of road segments. The second line of the pair gives r numbers, representing the amount of fuel it takes to travel between stations.

Output

For each road configuration, give the minimum cost (in dollars) of travelling the road.

Sample Input	Sample Output
5 10 4 5 4 2 9 5 4 5 5 5 5 10 4 1 2 3 4 10 4 8 2 8 2 10 14 8 1 3 5 6 9 7 2 8 1 2 5 6 7	61 15 0 10 127

Problem 5 – Drop it like it's full

Calvin mows lawns. While mowing, he bags the clippings from the grass and would like to know the positions on the lawn when the bag of clippings will be full. To simplify his calculations, Calvin has made some strict rules about how he will mow lawns:

1. Calvin will only mow rectangular lots.
2. Calvin mows in straight lines, makes perfect corners and mows each point in the lawn exactly once.
3. Each lot must have dimensions which are multiples of the width of the mower (so he never has to mow half-rows, etc).
4. Calvin will only mow the lawn starting at the top left-hand corner (the northwest corner) and proceeding in a clockwise direction. In doing so, he spirals from the outside of the lawn inwards towards the centre.
5. The mower's bag has a capacity S. Calvin has trained himself to push the mower at the right speed so that the bag is filled by mowing a line whose length is S times the width of the mower in length.
6. Initially, Calvin's bag is empty.

With these calculations, Calvin needs to know the positions in the lawn where the bag will become full. To represent this, you will draw the lawn as a grid and mark the positions of the lawn where the bag is full.

Input

The first line gives `n`, the number of tests in the input. The next `n` lines consist of three positive integers, `W`, `L` and `S`. `W` is the width of the lot, `L` is its length, and `S` is the size of the bag.

Output

For each case, output the line "`Lawn X:`", where `X` is the number of the lawn, starting at 1. Then output a grid of size `W`-by-`L`. In each position, place a period (`.`) if the bag is not filled at that position, and a capital letter x (`X`) if the bag is filled at that position. After the grid is printed, print a blank line.

Sample Input	Sample Output
3 5 5 3 4 6 3 3 10 4	Lawn 1: ..X.. ..X.X XX... ...X. .X..X Lawn 2: ..X. ..X. XX.X .... ..X. X..X Lawn 3: ... ..X .X. X.. ... ..X .X. X.. X

Problem 6 – Hands

As part of a program to visually display an analog clock, you are responsible for converting the time (given in `hours:minutes`) into two angles representing the positions of each of the two hands.

Angles are measured starting from the 12 o'clock position: 0 degrees would represent a hand pointing up, while 270 degrees would represent a hand pointing to the left.

Your angles should have at most one decimal digit. You may assume that each hand moves evenly through its turns: for example, at half past the hour, the hour hand is exactly half way between two hours.

Input

The first line gives the number of test cases. Each of the next n lines is of the form `xx:xx` where each `x` is a digit. This represents a valid time. The first two digits form the hour, a number between 1 and 12. The second two digits form the minutes, a number between 0 and 59.

Output

For each time, output on a single line two numbers (with one decimal place) representing the angle of the hour and minute hands.

Sample Input	Sample Output
3 12:00 03:00 03:14	0.0 0.0 90.0 0.0 97.0 84.0