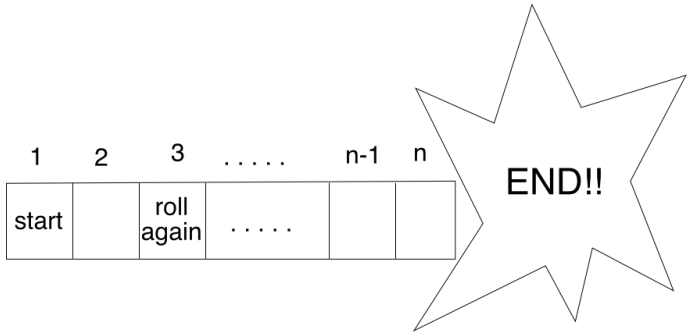
University of Manitoba
Open Programming Contest
September 29, 2007

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 those that are first.
3. All input should be read from standard input.

Problem 1 – Board Game Design

Jim is a designer for board games for children. Each board game is a path of spaces which a player's token may occupy. Jim is interested in knowing, for a particular board layout, the expected number of turns before a player will reach the end of the board.



Play begins with the player's piece on the start square and ends when the player passes the last square of the board. The player does not need to move exactly off the end of the board: any play that passes them beyond the last square is acceptable.

A single turn for a player consists of: rolling an *m*-sided die and moving the given number of spaces. If the new space is a "roll again" space, the player repeats the process. Otherwise, the player's turn is over.

Input

The configuration of one board consists of: an integer n (the number of spaces), followed by a sequence of length *n* consisting of 0s and 1s (representing the status of each of the spots on the board: 1 represents "roll again", and "0" does not) and finally an integer *m* (the number of sides on the die).
To avoid confusion for young players, the start space (the first entry in the sequence) is never a "roll again" space. A roll of the die produces one of the values from 1 to *m*, each with equal probability. For all configurations, $n \leq 80$ and $2 \leq m \leq n$.

The input to the problem consists of several game board configurations.
The final line of the input file will be "0", which should not be processed.

Output

The output should consist of the expected number of moves to reach the end of the game board configuration. Output should be formatted to eight decimal places (trailing zeroes may be omitted, except in the case of an integer value, which should be printed with one trailing zero; see below).

Sample Input	Output for Sample Input
5 00000 2 5 01111 3 10 0110010110 4 0	3.5625 1.0 2.65832901

Problem 2 – Writing Cheques

In order to prevent forgery, the amounts on mass-produced cheques are usually printed in a special format such as the following:

```
$****23,865.42
```

Write a program which will accept two parameters: a positive integer *C* representing an amount of money in cents, and an integer *N* between 5 and 15 which specifies the exact number of characters which must be used to represent that amount.

You may assume that *C* is small enough to fit into a standard 32-bit signed integer. Your program should write an amount in the format illustrated above, and which follows these rules:

1) There must be exactly two digits to the right of the decimal point, and at least one digit to the left of the decimal point.
2) There must be a comma (,) after every third digit, starting at the decimal point and moving to the left, provided that there is at least one digit to the left of that comma.
3) There must be no leading 0 digits, except as required by rule 1.
4) The first character in the string must always be a dollar sign ($).
5) There must be no blanks in the string. Any positions not otherwise filled must contain an asterisk (*).
6) If the rules above cannot be followed (because *C* is too big or *N* is too small), the entire string must be filled with question marks (?).

Input

The input consists an arbitrary number of cases, each on a separate line, followed by 0 on a single line. This should not be processed. Each case has two numbers: *C* and *N*.

Output

Output the formatted string for each case on a separate line.

Sample Input	Output for Sample Input
7 6 12345 10 1234567890 15 1234567 9 0	$*0.07 $***123.45 $*12,345,678.90 ?????????

Problem 3 – Sum to a Target

Write a program, which, given a list of up to 30 integers and a target, will find (and print) a subset of the numbers whose total is that sum if one exists or that will indicate that none exists, otherwise.

For example, for the list: 5, 13, 23, 9, 3, 3 and target 28, your program should find 5,23.

However, if more than one sum exists, you are required to find the sum which uses the largest possible elements from the set. For example, for the list 1, 2, 3, 4, 6 and target 11, the correct answer is 1,4,6 and not 2,3,6 (as 4 is greater than 3).

Input

Input consists of an arbitrary number of sets. Each set is given on one line and in the order: *N* (length of the list), followed by *N* values, followed by potential sum values *S*>0. The set may contain duplicates.

A value of *N*=0 with no following values causes the program to terminate. This input should not be processed.

Output

For each set, output on one line the subset which sums to the target and, if several exist, the subset with the largest element from the set. Output your set in increasing order or elements. If no set exists, print `**NO SET**` on the line.

Sample Input	Sample Output
5 1 2 3 4 6 11 6 5 13 23 9 3 3 28 2 1 2 10 0	1 4 6 5 23 **NO SET**

Problem 4 – King Movement

In the game of chess, a king can move one square in any direction (vertically, horizontally, diagonally). Suppose that we have an infinitely large chessboard with a single king. How many paths of *n* steps can this king move so that it ends up in its original square? Your task is to write a program that computes the number of possible paths for a particular *n*.

Input

The first line of the input is an integer *m* ($1 \le m \le 10$) followed by m test cases. Each test case consists of one line containing one number, *n* ($1 \le n \le 12$).

Output

For each test case, print one line containing the number of possible paths. Note that the solution will fit into a signed 32-bit integer.

Sample Input	Sample Output
3 1 2 3	0 8 24

Problem 5 – Minesweeper Predictor

Minesweeper is a popular time-waster. The goal of the game is to find where all mines are located on an M-by-N field.

The game shows a number in a square which tells you how many mines there are adjacent to that square. Each square has at most eight adjacent squares. In the example below, the 4x4 field on the left contains two mines, each represented by a "*" character. If we represent the same field by the hint numbers, we end up with the field on the right:

```
*...        *100
....        2210
..*.        1*10
....        1110
```

Input

The input is a list of an unknown number of fields. Each field begins with its dimensions: n and m ($1 \le n,m \le 80$). The number of rows in the field is *n* and the number of columns is *m*.

After the dimensions, there are n lines of text, each of length *m*. This represents the fields. There are no spaces in the lines, and the only characters are '*' (mines) and '.' (no mines).

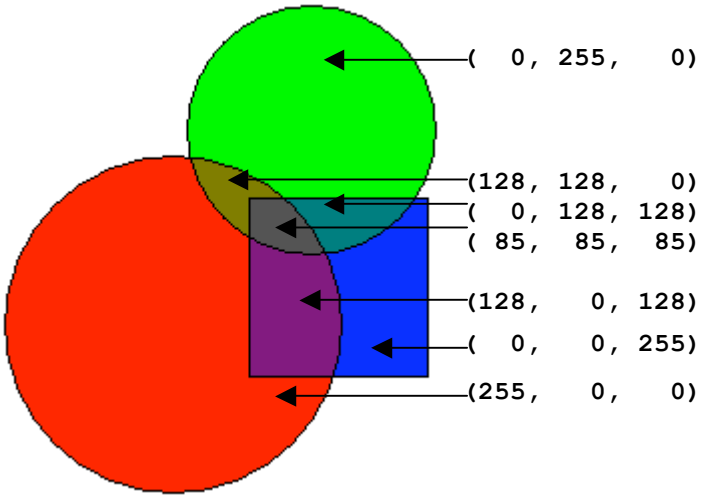
The last input is 0 0 representing the end of input. This should not be processed.

Output

For each field, print out a line containing "Field #X" where X is the number of the field, starting with 1. The next *n* lines should contain the hint field, where each '.' is replaced with a digit from 0 to 8 (inclusive). After the field, print out a blank line.

Sample Input	Output for Sample Input
4 4 *... .... .*.. .... 3 5 **... ..... .*... 0 0	Field #1 *100 2210 1*10 1110 Field #2 **100 33200 1*100

Problem 6 - Of Circles and Squares



A number of of circles and squares have been placed on an infinite stretch of a 2D plane. These circles and squares come in various sizes and colors. But when we place more than one object together, the color in the common region of those objects becomes a mixture of the original color. We define the color of each geometric object by the 8 bit value of the Red, Green and Blue components. For example, a circle drawn in pure red would have the color (255,0,0). The objects have **no borderlines: any point that is in or on the boundary of the region has the colour of that region**. Empty regions on the plane are always white (255,255,255). Given a number of such geometric objects and some points in the 2D plane, your task is to determine the color of the points.

The color of a point is computed as the average red, average green and average blue values of the geometric objects that this point falls into. If the point falls in the empty space, the color of the point would be white.

Input

The first line of input gives you the number of test cases T (T≤100) to follow. Each test case starts with two integers, R and P. R (R≤100) is the number of geometric objects and P (P≤100) is the number of query points. Then there will be R lines describing the objects. The description of each geometric object follows the “object_type px py length r g b” format. object_type is the type of the geometric object, which can be “CIRCLE” or “SQUARE”. px and py (-1000≤px,py≤+1000) are two integers giving you the x and y coordinate of the center of the circle or the lower left corner of the square depending on the object type. length (0≤length≤2000) is the radius of the circle or the length of a side if it is a square. The r, g and b (0≤r,g,b≤255) values are integers giving you the color of the object. The description of the query points starts after the description of the

objects. Each query point is denoted by an integer pair px and py (-1000≤px,py≤+1000) giving you the x and y coordinate of that point.

Output

For each test case, print "`Case x:`" first, where `x` is the case number. Then for each of the query points, print the color of that point in one line. All the colors are to be rounded to the nearest integer (for example: 128.3 will be rounded to 128, 128.5 will be rounded to 129 and 128.7 will also be rounded to 129). There must be a blank line separating two test cases. Please consult the sample input/output section for the exact format.

Sample Input	Output for Sample Input
2 1 2 SQUARE 0 0 5 0 0 127 6 6 0 5 3 3 CIRCLE 0 0 5 10 10 10 SQUARE 1 1 5 255 10 10 CIRCLE 5 5 3 0 0 0 1 1 6 5 8 5	Case 1: (255, 255, 255) (0, 0, 127) Case 2: (133, 10, 10) (128, 5, 5) (0, 0, 0)