UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS

MARATON SEMANA TECNOLOGICA

CATEGORIA AVANZADOS

UNIVERSIDAD DISTRITAL

5 de Noviembre de 2008

PROBLEMAS

Elaborado por: Hector Florez
Basado de www.acis.org.co

Problem 1. Christmas Trees

Source file name: tree.c, tree.cpp or tree.java
Input: tree.in
Output: standar output

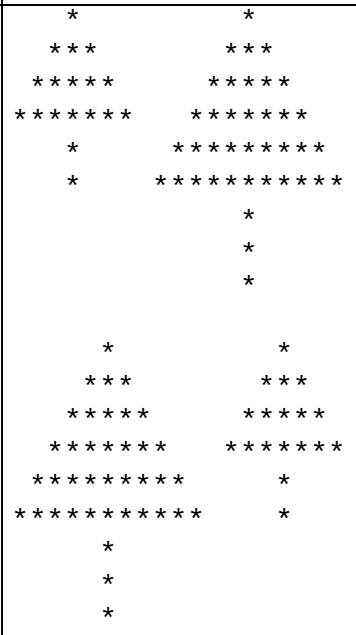
Christmas is always fun at the rebel home base. This year they are buying two Christmas trees and are placing them in elevated stands so that the tops of the trees are level if the trees are of different heights. Given the heights of the trees, you have to print a picture of the two Christmas trees.

Input

The input consists of several lines. Each line will contain two positive decimal integers no larger than 100. These integers will be separated by exactly one space. These integers represent the sizes of the trees; the size of the left tree is followed by the size of the right tree. The last line will contain two zeros, separated by one. This line is not to be processed; it merely signifies the end of the input.

Output

The output cases are to appear in the same order in wich they appear in the input. A full specification is not necessary; the examples will suffice, but here area few pointers: The height of the stem is the same as the input integer; the height of the branches is twice that. The tops of trees are level; the bottoms need not be. There is exactly one space between trees, wich is to say, between the trees is there is exactly one vertical column consisting only of spaces. All trailing spaces are trimmed from each line before printing; the last character of any line in the output file should not be a space. There should be exactly one blanck line following each output case.

Sample input	Sample Output
2 3 3 2 0 0	

Problem 2. Contest System Quality Assurance Tester

Source file name: quality.c, quality.cpp or quality.java
Input: quality.in
Output: standard output

Write a program to score a small, three-problem programming contest. Each input line contains six space-separated integers representing raw score data. The first three integers are in the range 0 . . . 100000. They represent seconds taken to solve the first, second, and third problems, respectively. Zero seconds indicates that a problem has not been solved. The last three integers are in the range 0 . . . 100, representing the attempts taken to solve the first, second, and third problems, respectively. Every failed attempt is penalized with 20 minutes, but only for problems that are eventually solved.

Each output line should begin with the string team, followed by a single space, the input line number, a colon, a single space, the number of solved problems, a comma, a single space, and the total number of seconds including penalties it took for the solved problems.

Sample input	**Sample Output**
0 777 0 4 1 1 1 1 1 1 1 1	team 1: 1, 777 team 2: 3, 3

Problem 3. Odd or Even

Source file name: odd.c, odd.cpp or odd.java
Input: odd.in
Output: standar output

There are several versions of Odd or Even, a game played by competitors to decide random issues. In one of the versions, for two players, the game starts with each player calling either odds or evens. Then they count to three (some people chant “Once, twice, three, SHOOT!”). On three, both players hold out one of their hands, showing a number of fingers (from zero to five). If the fingers add to an even number, then the person who called evens wins. If the fingers add to an odd number, then the person who called odds wins.
John and Mary played several games of Odd or Even. In every game John chose odds (and, Mary chose evens). During the games each player wrote down, in small cards, how many fingers he/she showed, using one card for each game Mary used blue cards, John used red cards. Their objective was to be able to re-check the results later, looking at the cards for each game. However, at the end of the day John dropped the deck of cards, and although they could separate the cards by color, they are now out of order. Given the set of numbers written on red cards and on blue cards, you must write a program to determine the minimum number of games that Mary certainly won.

Input

The input contains several test cases. The first line of a test case contains an integer N representing the number of games played ($1<N<100$). The second line of a test case contains N integers X_i, indicating the number of fingers shown by Mary in each of the games ($0<=X_i<=5$, for $1<=i<=N$). The third line of a test case contains N integers Y_i, indicating the number of fingers shown by John in each of the games ($0<=Y_i<=5$, for $1<=i<=N$). The end of input is indicated by $N = 0$.

Output

For each test case your program must write one line, containing one integer, indicating the minimum number of games that Mary certainly won.

Sample input	Sample Output
3 1 0 4 3 1 2 9 0 2 2 4 2 1 2 0 4 1 2 3 4 5 0 1 2 3 0	0 3

Problem 4. Making Money

Source file name: money.c, money.cpp or money.java
Input: money.in
Output: standar output

A trick sometimes used by parents to teach their children the value of money is to give then a penny - just a penny! - and the promise that for each day they don't spend it, the parent will double it. All students of computing know that long before a month has elapsed without spending a cent, the parents will not likely be able to make good on their promise. 100-percent compound daily interest on an investment is, of course, unattainable in normal financial dealings, but we are all continually reminded of the power of compound interest, even with the relatively low interest rates available today.

But exactly how much money can be made with compound interest? Assume, for example, an initial investment of $100.00 (US or Canadian), an annual interest rate of 6.00 percent, and that interest is compounded monthly. That is, the interest earned during the preceding month is added to the principal at the end of the month. (For our purposes, we'll assume a month is exactly 1/12th of a year.)

At the end of the first month, the money will have earned 0.5 percent interest (1/12th of 6.00 percent), or $0.50. This is added to the $100.00 invested, so that during the next month, interest is paid on $100.50. During the next month another 0.5 percent interest is earned, which is exactly $0.5025. We will assume that the bank, being conservative, will not pay any interest less than $0.01, so our investment is credited with an additional $0.50 at the end of the second month, for a whopping total of $101.00. Continuing in the same manner, at the end of 12 months our investment will total $106.12, $0.12 more than simple 6.00 percent interest for a year with no compounding. Given an amount P to be invested for a year with I percent interest, compounded C times during the year at equal intervals, what is total return on the investment?

Input

There will be multiple cases to consider. The input for each case is a single line containing the initial investment amount, P, given in dollars and cents (but no fractional cents, and no larger than $100,000.00), the annual interest rate (I) given as a real number with two fractional digits representing a percentage, greater than zero but less than 100, and the number of compounding intervals per year (C), an integer between 1 and 365. The last case will be followed by a line containing "0.00 0.00 0".

Output

For each input case, display the case number (1, 2, . . .), the initial investment (P), the annual interest rate (I), the number of compounding intervals per year(C), and the value of the investment at the end of a year. Your output should follow the format shown in the examples below.

Sample input	Sample Output
100.00 6.00 1 100.00 6.00 12 1000.00 6.00 12 0.00 0.00 0	Case 1. $100.00 at 6.00% APR compounded 1 times yields $106.00 Case 2. $100.00 at 6.00% APR compounded 12 times yields $106.12 Case 3. $1000.00 at 6.00% APR compounded 12 times yields $1061.63

Problem 5. Ball Toss

Source file name: toss.c, toss.cpp or toss.java
Input: toss.in
Output: standar output

Classmates stand in a circle facing inward, each with the direction left or right in mind. One of the students has a ball and begins by tossing it to another student (It doesn't really matter which one). When one catches the ball and is thinking left, he throws it back across the circle one place to the left (from his perspective) of the person who threw him the ball. Then he switches from thinking left to thinking right. Similarly, if he is thinking right, he throws the ball to the right of the person who threw it to him and then switches from thinking right to thinking left.
There are two exceptions to this rule: If one catches the ball from the classmate to his immediate left and is also thinking left, he passes the ball to the classmate to his immediate right, and then switches to thinking right. Similarly, if he gets the ball from the classmate to his immediate right and is thinking right, he passes the ball to the classmate to his immediate left, and then switches to thinking left (Note that these rules are given to avoid the problem of tossing the ball to oneself).
No matter what the initial pattern of left and right thinking is and who first gets tossed the ball, everyone will get tossed the ball eventually! In this problem, you will figure out how long it takes. You'll be given the initial directions of n classmates (numbered clockwise), and the classmate to whom classmate 1 initially tosses the ball (Classmate 1 will always have the ball initially).

Input

There will be multiple problem instances. Each problem instance will be of the form n k t1 t2 t3 … tn where n (2 <= n <= 30) is the number of classmates, numbered 1 through n clockwise around the circle, k (> 1) is the classmate to whom classmate 1 initially tosses the ball, and ti (i = 1, 2, … , n) are each either L or R, indicating the initial direction thought by classmate i (n = 0 indicates end of input).

Output

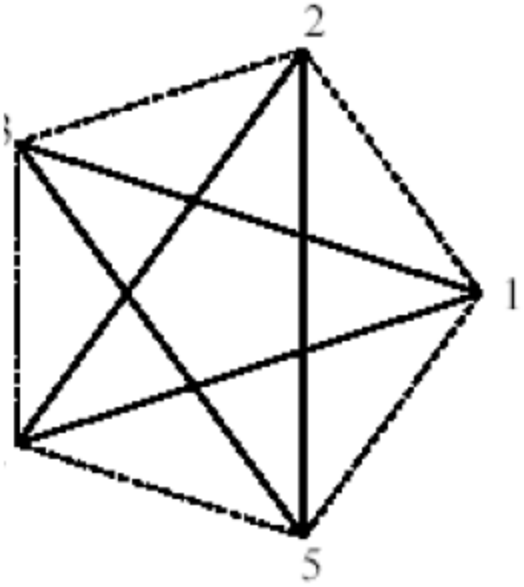
For each problem instance, you should generate one line of output of the form:
Classmate m got the ball last after t tosses.
where m and t are for you to determine. You may assume that t will be no larger than 100,000. Note that classmate number 1 initially has the ball and tosses it to classmate k. Thus, number 1 has not yet been tossed the ball and so does not switch the direction he is thinking.

Sample input	Sample Output
4 2 L L L L 4 3 R L L R 10 4 R R L R L L R R L R 0	Classmate 3 got the ball last after 4 tosses. Classmate 2 got the ball last after 4 tosses. Classmate 9 got the ball last after 69 tosses.

Problem 6. Making Stars

Source file name: stars.c, stars.cpp or stars.java
Input: stars.in
Output: standard output

Everyone can make stars. A five-pointed star is usually drawn as shown (by the solid lines) in figure 1. Here the points of the star have been labeled 1 through 5. The usual procedure for drawing the star is to connect point 1 to point 3, point 3 to point 5, point 5 to point 2, point 2 to point 4, and finally connect point 4 to point 1. This is all done without lifting your pencil from the paper.

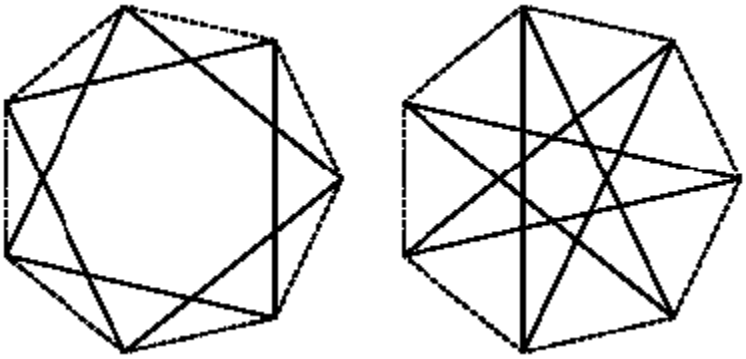


The formal procedure for drawing an n-pointed star (without lifting the pencil) is to begin with an n-sided polygon (shown with the dashed lines in figure 1). Pick any polygon vertex as a starting point, and then draw consecutive lines from one polygon vertex to another until you return to the starting vertex, skipping the same number of vertices, in the same direction (clockwise or counterclockwise) each time. A requirement is that we must skip at least one vertex between every pair of connected vertices, or else we just wind up tracing the edges of the polygon. Likewise, we must also skip fewer than n–1 vertices.

For example, with the 5-pointed star, we might start with vertex 1, skip vertex 2, and draw a line to vertex 3. We then skip vertex 4, and draw a line to vertex 5. Skipping vertex 1, we draw a line from vertex 5 to vertex 2. Skipping vertex 3, we draw a line from vertex 2 to vertex 4. Finally, we skip vertex 5, and draw a line from vertex 4 to vertex 1, completing the star.

We could also skip two vertices when drawing the 5-pointed star. This would result in connected vertex 1 to vertex 4 (skipping 2 and 3), then connecting vertex 4 to vertex 2 (skipping 5 and 1), then vertex 2 to vertex 5 (skipping 3 and 4), then vertex 5 to vertex 3 (skipping 1 and 2), and finally vertex 3 to vertex 1 (skipping 4 and 5). This is exactly the same star produced by skipping just one vertex. That is, the same vertices are connected by lines as in the earlier star.

It is not, however, always true that skipping different numbers of vertices will yield the same star. For example, there are two different 7-pointed stars that can be drawn, as shown in figure 2.



The question you are to answer in this problem is how many different stars with n points can be drawn in this manner?

Input

There will be multiple cases to consider. The input for each case is a single integer n, between 5 and 500, that specifies the number of points in a star. The last case will be followed by the integer 0.

Output

For each input case, display the case number (1, 2, . . .), the number of points (n), and the number of unique stars that can be drawn without lifting your pencil. Your output should follow the format shown in the examples below.

Sample input	Sample Output
5 6 7 0	Case 1, n = 5, unique stars = 1 Case 2, n = 6, unique stars = 0 Case 3, n = 7, unique stars = 2