NEW ZEALAND PROGRAMMING CONTEST 1998

Problem A — A Nice Number — 5 points

Print the smallest positive whole number whose digits are all different and which is exactly divisible (with no remainder) by 2, 3, 4, 5, 6, 7, 8 and 9

EXAMPLE
Input

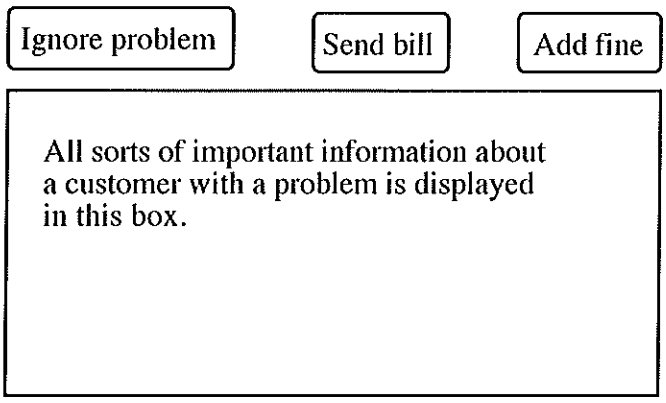
(There's no input)

Output
```
421
```

(Actually the number isn't 421 but this is to show that the output, on the screen, must be a single number starting at the left of the screen. Many of the answers to the problems are of this, or a similar, form. Do NOT try to "improve" this output by writing "Number = 421" or similar; this will be regarded as a wrong answer).

Problem B **Button Placement** 5 points

Suppose you are designing a program which will display some information on the screen, and the user has to respond by using the mouse to click one of the buttons on the screen, as in the following:



To make the program more user-friendly, you have to place the buttons so that they do not look crowded (assume you are using a primitive language which does not place the buttons for you). One way of placing them is shown above - the outer buttons line up with the edges of the box, and the buttons between them are placed so the spacing between the buttons is exactly the same (to within one pixel - for example, 130 pixels between some buttons and 129 between the others).

For this problem you will be given several sets of data which describe a situation like the one shown (but possibly with more buttons above the box), and your program must provide information about where the buttons must be placed, in the horizontal direction. The information you will be given is the number of buttons (between 2 and 5), the width of the box, and the width of each button. You must give the placement of the left side of each button, in pixels, relative to the left edge of the box (so the leftmost buttons will have a placement of 0). The number of pixels between the right and left edges of adjacent buttons must all be the same if possible; if it is not possible to make all the spacings the same, then the spacings must differ by no more than 1 pixel, with the widest spacings all as far left as possible.

Input will be from standard input (ie the keyboard or a redirected file) and will be two lines for each placement problem. The first line will contain two numbers separated by one or more spaces. The first number gives the number of buttons above the box, N, and the second number gives the width of the box in pixels, W. N will always be greater than 1 and less than 6. The next line contains N numbers, separated by one or more spaces, giving the width of each button. The sum of these numbers will always be less than W - N. The end of the placement problems will be indicated by a line containing two 0's, ie N = W = 0.

Output, which must be written to standard output (the screen), must be one line for each placement problem in the input, giving the distance of the left edge of each button, in pixels, from the left edge of the box. The numbers must be separated by a single space

EXAMPLE

Input

```
3 576
143 82 92
2 576
187 187
0 0
```

Output

```
0 273 484
0 389
```

Problem C **Report** 5 points

For this problem you must produce a simple report.

Input will be from standard input (ie the keyboard or a redirected file) and will be a set of lines containing sales data. Each line will contain two numbers relating to an item, the first giving the cost price in dollars and cents, and the second giving the sales price in dollars and cents. The prices will all be less than 1000000.00 and greater than or equal to 0.00. The prices will be followed by the name of the item sold, which will always be less than 20 characters. The numbers, and the name, are all separated by one or more spaces. Each line will be less than 60 characters long. There could be any number of lines for the report (tens of thousands). The input is terminated by a line containing two zeros.

Output, which must be written to standard output (the screen), must be a line of headings and a line of totals, as shown below, and between them one line for each line in the input. Each of these lines must start with the name of the item, left justified (ie no leading spaces), followed by the cost price, then the sales price, then the profit on the item which is worked out as:

profit = sales price - cost price.

Counting the first character on each line as column 1, the decimal point of the cost price must be in column 30, the decimal point of the sales price in column 40, and the decimal point of the profit in column 50. The total line must be like the other lines in the report, with an item name of TOTAL, and the amount in each column being the sum of the amounts of the data in that column. You may assume all the totals are less than 1000000.00 and greater than -1000000.00. The headings must line up with the decimal points as shown below.

EXAMPLE

Input

```
11.50 13.00 Shovel
59.00 103.00 Wheelbarrow
15.50      0.00  Raincoat
0.00 951376.41 Gold Nuggets
23167.50   4671.00           Mechanical Scrubber
0 0
```

Output

```
Item Name                 Cost     Sales    Profit
Shovel                   11.50     13.00      1.50
Wheelbarrow              59.00    103.00     44.00
Raincoat                 15.50      0.00    -15.50
Gold Nuggets              0.00 951376.41 951376.41
Mechanical Scrubber   23167.50   4671.00 -18496.50
TOTAL                 23253.50 956163.41 932909.91
```

Problem D — Caesar Cipher — 5 points

A simple way of concealing information is to use a cipher to transform each letter into another one; a way of doing this is called a cipher. The simplest cipher was used by Julius Caesar more than two thousand years ago; in this "Caesar cipher", each letter is changed into one a fixed number ahead - for example, with a shift of 3, a is changed to d, b to e, c to f and so on. The letters wrap around after z so that w is changed to z, x is changed to a, y to b and z to c.

Input will be from standard input (ie the keyboard or a redirected file) and will be in pairs of lines, where the first line contains a number, giving the amount of forward shift (always > 0 and < 26), and the next is a line of text to be changed using the cipher (lines will always be less than 60 characters). The input is terminated by a line containing a single zero.

Output, which must be written to standard output (the screen), must be a the lines of text which are changed using the Caesar cipher. Make sure that only the letters in the line are changed - spaces and punctuation are to be unaffected. Also, uppercase and lowercase form must be preserved.

EXAMPLE

Input

```
3
A simple way of concealing information is to use a cipher
10
to transform each letter into another one; a way of doing
25
this is called a cipher.  The simplest cipher was used by
1
Julius Caesar more than two thousand years ago.
0
```

Output

```
D vlpsoh zdb ri frqfhdolqj lqirupdwlrq lv wr xvh d flskhu
dy dbkxcpybw okmr voddob sxdy kxydrob yxo; k gki yp nysxq
sghr hr bzkkdc z bhogdq.  Sgd rhlokdrs bhogdq vzr trdc ax
Kvmjvt Dbftbs npsf uibo uxp uipvtboe zfbst bhp.
```

Problem E **Depreciation** 5 points

Computers lose value very fast. Just when you thought you had the latest and best, two weeks later another model is announced and your computer is not nearly as valuable! Your task in this problem is to write a program which will compute just how much less valuable your computer is, as the weeks pass, assuming a constant rate of loss. You will be given the original value of the computer, and the percent by which it loses value ("depreciates") each week. For example, if the original value was $5000 and the rate of depreciation is 2%, then after one week it has lost $100, so its value is $4900. Next week it loses 2% of $4900, ie $98, so its value is $4802. Next week it loses $96.04, so its value is $4705.96. The next week it loses $94.1192, but because we can't deal in fractions of a cent, this is truncated to $94.11, so its value after four weeks is $4611.85. Note that this is not the same as you would get by simply taking 4 x 2% off $5000 - this would give $4600.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of a number of depreciation scenarios. Each will be described by three numbers, the first giving the original value of the item in dollars and cents, the second giving the percentage depreciation per time period, and the third giving the number of time periods the depreciation must be computed over. The original value will always be less than $1,000,000. The percentage depreciation will always be a whole number less than 100 and greater than 0. The number of time periods will always be less than 500. The output will be terminated by a line containing three 0's.

Output, which must be written to standard output (the screen), must be one number for each depreciation scenario, giving the final value of the item, in dollars and cents (ie with two decimal places). Note that the amount depreciated must be computed for each time period and then truncated to give whole cents, as shown in the calculation above.

EXAMPLE

Input

```
5000.00 2 4
44281.15 10 10
0 0 0
```

Output

```
4611.85
15439.91
```

Problem G **Golf** 15 points

An "intelligent" golf ball has been invented, which will mean that golfers no longer need to keep scorecards. This golf ball has built-in electronics (very rugged), so that every time it is hit, it sends a message to the club house giving its ID. Furthermore, every one of the 18 holes in the golf course is wired so that when a ball drops into it, the hole sends a message giving its number and the ball's ID. According to the rules of golf, each player must have a single ball, which they must hit (possibly taking several hits) into each hole. The number of hits each player takes for each hole can now be counted automatically, and the outcome of a tournament decided before the golfers reach the clubhouse (the winner is the one with the fewest hits).

You must write a program which will analyse the messages received at the clubhouse and compute the total number of hits each ball has received. Players must play each hole, from 1 to 18, in order; thus, for each ball, you expect a sequence of hits followed by a hole, which must be first number 1, then 2, then 3 and so on up to 18. If a hole is played out of order, or the ball is simply carried to the next hole and dropped in, the round for that ball is invalid. A final total can only be computed for those balls which have been hit into hole 1, then 2, etc up to hole 18. Of course, the messages are sent randomly as the various balls are hit in different parts of the golf course, so the input stream of messages will closely resemble a random list of ID numbers, with hole numbers scattered randomly throughout the list.

Input will be from standard input (ie the keyboard or a redirected file) and gives the data for a single period of time on the golf course. The input will consist of lines of numbers separated by spaces, with no more than 80 characters on each line. The numbers will either be ball ID numbers (between 100 and 999) or hole numbers (from 1 to 18). No more than 100 balls will be used. Each hole number will be followed by the ID of the ball which is in the hole. The input will be terminated by the number 0.

Output, which must be written to standard output (the screen), must list the ball ID's you have found in the input (in increasing numeric order), followed by a space then either the total number of hits recorded for that ball, or the number 0. You must print 0 if the round was invalid for that ball - ie if the hole numbers recorded for that ball do not follow the strict sequence 1, 2, ..., 18, or if there was no hit recorded for the ball between two successive hole numbers. Any hits and holes after hole 18 has been played should be ignored - the player must have been simply practising.

EXAMPLE:

Input

```
100 111 222 100 222 100 111 1 111 100 222 111 222 2 111  100 1 222 111
100 111 222 3 111 100 222 100 111 2 222 111  100 4 111 222 100 100 111
222 5 111 6 100 3 222 111 222 100  100 111 222 100 222 100 111 6 111 100
222 111 222 7 111  100 4 222 111 100 111 222 8 111 100 222 100 111 5 222
111  100 9 111 222 100 100 111 222 10 111 1 100 6 222 111 222  100 111
222 100 222 100 111 11 111 100 222 111 222 12 111 100 7 222 111 100 111
222 13 111 100 222 100 111 8 222 111 100 14 111 222 100 100 111 222 15
111 2 100 9 222 111 222 100 111 222 100 222 100 111 16 111 100 222 111
222 17 111 100 10 222 111 100 111 222 18 111 100 222 100 111 11 222 111
100 4 111 222 100 100 111 222 5 111 3 100 12 222 111 222 100 100 111 222
100 222 100 111 11 111 100 222 111 222 12 111 100 13 222 111 121 111
222 13 111 100 222 100 111 14 222 111 100 14 111 222 100 100 111 222 15
111 4 100 15 222 111 222 100 111 222 100 222 100 111 16 111 100 222 111
222 17 111 100 16 222 111 100 111 222 18 111 100 222 100 111 17 222 111
100 4 111 222 100 100 111 222 5 111 18 100 18 222 0
```

Output

```
100 0
111 32
121 0
222 53
```

Problem H **Lists of words** 15 points

For this problem you will be given a set of words, and you must output the word in a given position when they are arranged in increasing alphabetical order. For examples, if the words are "cat", "ant", "ewe", "bee", "dog", and you want the 2nd in alphabetic order, then the word "bee" must be output.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of a number of problem sets. The data for each problem starts with a line containing a single number, k, which is the position of the word wanted when the words are in alphabetic order. This line will be followed by from 1 to 100 lines containing the words, each word being at most 20 characters in length and entirely in lowercase letters. There could be duplicate words in the list; if so, the term "increasing order" should be interpreted to mean that all the words which are the same are listed one after the other. The list of words will be terminated by a line consisting of the single character #. The end of the problem sets will be given by a line containing the number 0 (ie k = 0 for a problem set).

Output, which must be written to standard output (the screen), must be one word on a separate line for each problem set, the word requested as above.

EXAMPLE
Input

```
2
cat
ant
ewe
bee
dog
#
2
cat
cat
cat
#
0
```

Output

```
bee
cat
```

Problem I **Summing Digits** 15 points

Given any number, the sum of its digits characterises the number, but there is a lot of duplication. For example, the numbers 431, 17, 1001200201001 and 8 all have the digit sum of 8. For this problem you need to find the smallest number whose digits sum to a given number.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of lines each with a single whole number, n, which could be positive or negative. If n is positive, you must find the smallest number whose digits sum to n (n will be no more than 81). If n is negative, you must find the smallest number, all of whose digits are different, and whose digits sum to abs(n). If n is negative it will be no smaller than -45. The input will be terminated by a line consisting of a single 0.

Output, which must be written to standard output (the screen), must be the solution number for each non zero number in the input.

EXAMPLE

Input

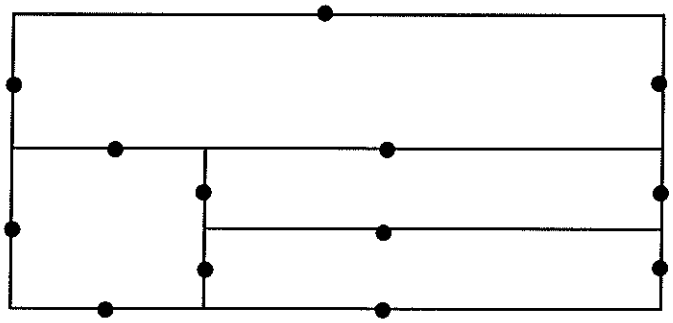
```
8
21
-21
0
```

Output

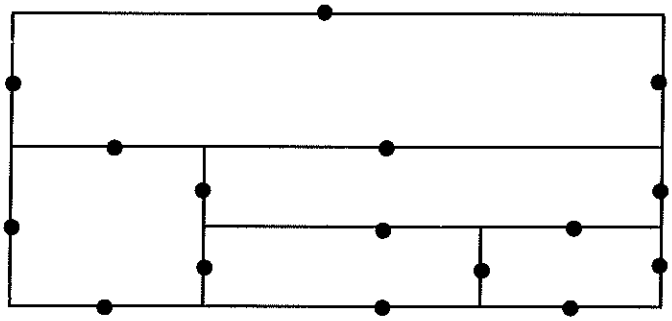
```
8
399
489
```

Problem J **Smart House Wiring** 15 points

A firm is constructing "Smart Houses" which have single processors imbedded in each internal and external wall. They have developed a very simple network, which depends on a single wire joining all the processors, running from a starting processor to a finishing one, crossing the wall at each of the other processors, not crossing any wall at any other place, and not crossing itself (in the plan). For example, a possible wiring scheme is shown for the house below:



They have been surprised to find that no such wiring scheme is possible for the following house plan:



This is a classic topological problem with an easy answer: a wiring scheme is possible if the house has no more than two rooms with an odd number of sides. You must write a program to determine whether a wiring scheme is possible, given the coordinate of the corners of the house. You may assume every wall is parallel to the x or y axis.

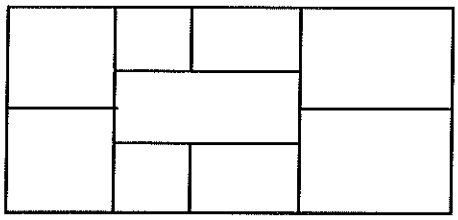
Input will be from standard input (ie the keyboard or a redirected file) and will consist of several house plans. Each house plan is given by a series of lines which describe the walls parallel to the x-axis, in increasing y-order. The y-coordinate of this wall is given (an integer greater than 0), followed by the x-coordinates (integers greater than 0) of points where walls parallel to the y-axis meet this wall, given in increasing x-order. The list of x-coordinates is terminated by a 0. The input is terminated by a 0 (ie an empty house plan). Two separate walls will never have the same x-coordinate or y-coordinate, i.e. the situation on the right below will not occur. The houses pictured above are given in the examples.

Output, which must be written to standard output (the screen), must be the word "Yes" if a wiring scheme is possible for this house plan, otherwise the word "No".

EXAMPLE

Input	**Output**
1 1 3 7 0 3 3 7 0 5 1 7 0 9 1 7 0 0 1 1 3 5 7 0 3 3 5 7 0 5 1 7 0 9 1 7 0 0 0	Yes No

Separate lines,
same y-coord



Separate lines,
same x-coord

WILL NOT OCCUR

Problem K **Table Headings** 15 points

In Problem C you had to print out a table with numbers arranged in columns, and with fixed heading on the columns. For this problem you must print out the same sort of table, but this time the headings will not be fixed but will be supplied together with the data and they could be quite long. In fact you could have up to four lines for the headings, and you must break the headings at a gap between words. See the example below.

Input will be from standard input (ie the keyboard or a redirected file) and will be a series of tables to be printed out. Each table starts with a number giving the number of columns, N, followed by a line with N numbers separated by spaces, giving the width of each column in characters. After this comes N lines, the first of which contains the heading for the first column, the next the second column, and so on. The data for the body of the table follows, which will consist of N-1 real numbers followed by a name, exactly as in Problem C. The table data is terminated by N-1 zeros, and may be followed by additional tables. The end of the input is a line containing a single zero, i.e. a table with 0 columns. Each input line will be less than 60 characters.

Output, which must be written to standard output (the screen), must be the table with each column headed by its heading. The heading may consist of several lines, and the line given must be broken up so it fits into the corresponding column width - 1 (to allow for a gap between headers). The breaks must be at space characters, and the first line, second line etc of the heading must be as long as possible, so the number of lines used by the headings is a minimum. In the first column, everything including the heading parts must be left justified, and in the columns everything is right justified. The last line of each heading must be the line above the data. There is no total line or profit column in this problem. You can assume that it is always possible to break the heading as specified so that it fits into at most four rows; in particular, every word in the header will be shorter than the column width. There must be a blank line between every table.

EXAMPLE

Input

```
3
20 15 15
Name of the item found by Police at the scene
Estimated Value
Price being charged by the suspect
11.50 13.00 Shovel
59.00 103.00 Wheelbarrow
15.50      0.00  Raincoat
0.00 951376.41 "Gold" Nuggets
23167.50   4671.00           Mechanical Scrubber
0 0
0
```

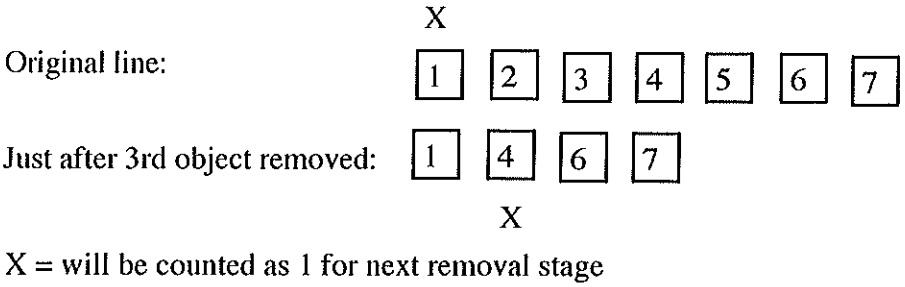
Output

```
Name of the item                    Price being
found by Police at        Estimated charged by the
the scene                     Value        suspect
Shovel                        11.50          13.00
Wheelbarrow                   59.00         103.00
Raincoat                      15.50           0.00
"Gold" Nuggets                 0.00      951376.41
Mechanical Scrubber        23167.50        4671.00
```

Problem M **Josephus Problem** 50 points

A classic problem in this sort of contest, which dates from several thousand years ago (it is rumoured to have first appeared in the 1541BC New Zealand Programming Competition), is the "Josephus" problem. The basic motivation for this is that one person or thing has to be selected from a group, in a random manner. Because randomness was not well understood a few thousand years ago, a mechanical process which is hard to analyse was used for the selection. You must write a computer program which will work out, for each Josephus problem, the thing chosen.

A number J is first chosen - usually, this is less than the total number of things to be selected from, but it could be greater. Then the objects from which the selection must be done are arranged in a line, and counting is started from the first one, which is counted as "1", the next one is "2", and so on. When the thing numbered "J" is reached, it is removed from the line, everything after it is moved up one place to fill in the gap, and counting is started again, taking the thing which has just moved into the gap as "1". When counting reaches the end of the line, it is continued from the beginning without a break - for example, if the last thing in the line has number "2", then the first thing in the line is counted again as number "3". For example, taking the 7 objects below, and taking J as 5, then the situation is shown after three objects have been removed:



X = will be counted as 1 for next removal stage

The process stops when there is one thing left, and this is the selected thing. In the case shown, it will be the thing numbered 6. The order the things are removed is 5, 3, 2, 4, 7, 1.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of lines with two positive whole numbers on each line, both more than 1 and less than 10,000. The first number gives the number of objects in the original line and the second number gives the Josephus counting distance, J. The input will be terminated by a line consisting of a two 0's.

Output, which must be written to standard output (the screen), must be one number for each line of input, giving the number, in the original line, of the selected item.

EXAMPLE

Input

```
7 5
8 9
999 888
0 0
```

Output

```
6
8
755
```

Problem N **Snakes and Ladders** 50 points

A popular board game for children is called "Snakes and Ladders". The board has squares which are numbered from 1 to 100, and players have counters which start on the theoretical square 0. The players take turns at throwing a die with the numbers 1 to 6 on it, and each moves his or her counter forward the number of squares corresponding to the uppermost number on the die (the square they reach is found by adding the die number to the square number their counter is on). The first person to reach square 100 is the winner. The interest is caused by the fact that pairs of squares are connected together by "ladders" (which connect a lower-numbered square to a higher-numbered square) and "snakes" (which run from high to low). If a counter lands on the start of a snake or ladder (ie this is the square reached after throwing the die), then the counter is moved to the corresponding square at the end of the ladder or snake. Note that landing on the end square of a ladder or a snake has no effect, only the start square counts. Furthermore, there are some squares such that if a player's counter lands on them, then the player must either miss their next turn, or immediately throw the dice again for another turn, depending on what is written on the board. A miss-a-turn or extra-turn square is never the start or end of a ladder or snake. If a player is on square 95 or higher, then a die throw which takes them past 100 must be ignored - thus a player on square 99 must ignore all throws which are not 1. Actually, in many games, square 99 is the start of a long snake so this particular case doesn't arise, and furthermore throwing the die on squares 93 to 98 can be stressful.

Input will be from standard input (ie the keyboard or a redirected file) and will start with a set of less than 1000 die throws which you must use for all games, starting each new game with the first player "throwing" the first number in the set, the next player "throwing" the second number, and so on. This set of die throws will simply be a list of random numbers between 1 and 6, separated by single spaces, with not more than 80 characters on each line. It will be terminated by the number 0. After this set of die throws, there will be one or more game sets. Each game set is in three parts. The first part is a line containing a single number giving the number of players in the game. This will be more than 1 and less than 6. Then the board is described, in two parts. The first part lists the ladders and the snakes on the board, each ladder or snake being defined on a single line. Each is given by two numbers, from 1 to 99, separated by one or more spaces. The first number gives the start square, and the second number gives the end square; so it is a ladder if the first number is less than the second number, and a snake if the order is the other way. The snake/ladder definitions are terminated by a line containing two 0's. The second part of the board description gives the lose-a-turn/extra-turn squares. These are single numbers, one per line, defining the squares. If the number is negative, it is a lose-a-turn square, if positive an extra-turn square. The end of this set of descriptions, and of the game description, is given by a single 0. The end of all the game descriptions is given by a game with the number of players equal to 0.

Output, which must be written to standard output (the screen), must be one line for each game in the input, giving the number of the player who wins the game. Every game will determine a winner in fewer throws that those given at the start of the data.

EXAMPLE

Input

```
3 6 3 2 5 1 3 4 2 3 1 2 0
2
6 95
99 1
0 0
-3
98
0
2
3 99
6 90
0 0
0
0
```

Output

```
2
2
```

Problem O **Real numbers** 50 points

For this problem you will have to write a parser to analyse numbers expressed in Pascal real format. This problem will be no simpler for teams using Pascal (if any) because the size of the numbers involved will be too large to be handled by the Pascal version used in this contest.

Pascal requires that real constants have either a number including a decimal point, or a number followed by an exponent (which is the letter e or E followed by an integer). If a decimal point is included it must have at least one decimal digit on each side of it. As expected, a sign (+ or -) may precede the entire number, or the integer in the exponent, or both. Exponents may not include decimal points. Blanks may not be embedded within the real constant. Note that the Pascal syntax rules for real constants make no assumptions about the range of real values, and neither does this problem.

For example, 112.65 is a valid Pascal real constant, and it is the same number as all of the valid constants +1.1265E+2, 0.01265e4, 11265E-2 and +1126500.0e-6.

Your task in this problem is to process possible Pascal real constants, identify whether they are valid format, and rewrite them in a "normal form", which is a Pascal real constant written in the format ±d.nnn..E±mm... where d is a single nonzero digit, nnn... is the decimal part which has no trailing zeroes (unless it is a single zero) and mm... is a whole number with no leading zeroes unless it is a single zero (in which case the sign preceding it must be +); for example, +1.1265E+2 is the normal form for 112.65, and -1.0E+0 is the normal form for -1.0. The number zero cannot be represented in this form, so it has the special normal form +0.0E+0

Input will be from standard input (ie the keyboard or a redirected file) and will consist of lines each consisting of a possible Pascal real constant. Any exponents in the valid constants given will always lie between -1000000 and +1000000, and the numbers could be any size, except that the line will have a maximum of 60 characters. The input will be terminated by a line consisting of a single #.

Output, which must be written to standard output (the screen), must be one line for each line of input, giving either the normal form of the number if the Pascal real constant is valid, or the word "Invalid". The answer will never be will always be able to be expressed in 60 characters or less.

EXAMPLE

Input

```
1.2
1.
e-12
-6.5E
412345.67890e-00999999
7.6e+12.5
99
123x456.789
#
```

Output

```
+1.2E+0
Invalid
Invalid
Invalid
+4.123456789E-999994
Invalid
Invalid
Invalid
```

Problem P **Farmer Bob's Sprogs** 50 points

In a certain area of the country (beyond Erewhon) the fields are laid out in a totally regular grid, each field being square and exactly one hundred acres in area (so each side is about 600m), and oriented so the sides run North, South, East and West. Farms consist of a number of these squares, each of which is connected to the other squares in the farm by at least one side. Thus in the diagram below, the left-hand picture shows a valid farm while the other pictures are not valid farms.

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

Farmer Bob has a farm which consists of 17 fields, which he wishes to divide among his three sons. The fields cannot be subdivided, so each will get five and the remaining two will be left as isolated bush reserves. Farmer Bob found it a bit difficult to see how to make three valid 5-field farms from his farm, which is shaped like this:

```
  XXX X
 XXXXXXX
XX  X  X
  X  X
```

so he called you in to write a general program to solve the problem, with the idea that a lot of his neighbours will be wanting to do this too.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of data for several farms. Each data set starts with a line containing a single integer, giving N ($2 \le N \le 5$), the number of parts the farm must be split into. The size of each part must be (total size)/N, rounded down. This is followed by a series of lines describing the farm, by giving a series of East-West traverses, from the North end of the farm to the South. Each line consists of a string of 0 and 1, a 0 meaning the field is not part of the farm and a 1 meaning it is. No farm is bigger than 10 fields in any dimension. The file will be terminated by a line containing a single 0.

Output for each farm, which must be written to standard output (the screen), must start with the words "Farm Number ", followed by the number of the farm data set in the input, the first data set being number 1. After this must follow either the word "Impossible", if the farm (with a number less than N of fields removed, if necessary) cannot be split evenly, or the farm map with the 1's replaced by a,b,c etc to indicate the new farms being created, where a is used for all fields in the first farm, b for all fields in the second farm, and so forth (any fields left over remain as 1's). Any of the possible solutions will be marked correct. There must be a blank line after each farm map (or "Impossible").

EXAMPLE

Input

```
2
11111
3
11101000
01111111
11001001
01001000
0
```

Output

```
Farm Number 1
aabb1

Farm Number 2
Impossible
```

Problem Q **New Calendars** 50 points

The new Minister of Efficiency (your local MP) has just had a stroke of brilliance. He believes we can improve the output of the country by 14% (1/7) by moving to an 8-day week instead of a 7-day week. His logic that the weekly output of the country must rise by a factor of 1/7 is inescapable, especially as he has assigned you the lucrative contract of recomputing calendars for future years. You suspect he would be even more impressed by the efficiency gains with 9-day or 10-day weeks, but his advisers haven't pointed this out to him yet. The new day names will be the more efficient names day0, day1, day2 etc, and (whatever week length is chosen) the first day of the year 2000 will be called day0.

Your task is to write a program which will print out "months" for future years, given that weeks are not 7 days but 8,9,10 etc, where a "month" is a set of 30 consecutive days. You will be given a year (we will continue to use standard Gregorian years) and a day number in that year, and you must take this as the first of the month and print out the next 30 days, using the standard calendar format as shown below (but with bigger weeks and the new day names). Note that there is a single space between each day name, and the columns are left justified.

Remember that, in the standard Gregorian calendar we use, there are 365 days in a year except for leap years, when there are 366. Leap years are all years divisible by 4 and not 100, except that those divisible by 400 are leap years - thus 1900 was not a leap year, 1904, 1908 ... 1996 were leap years, 2000 will be a leap year, 2100 will not be a leap year, etc.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of lines containing three numbers, separated by one or more spaces. The first number on each line will be the number of days in the new week (between 7 and 12), the second number will be the year number (less than 1,000,000) and the third number will be the day of the year that the calendar starts.. The input will be terminated by a line consisting of three 0's.

Output, which must be written to standard output (the screen), must be a calendar in standard format (with wide weeks) as shown below.

EXAMPLE

Input

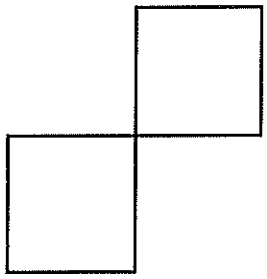
```
8 2000 1
9 4000 58
0 0 0
```

Output

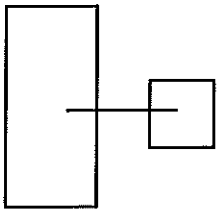
```
day0 day1 day2 day3 day4 day5 day6 day7
   1    2    3    4    5    6    7    8
   9   10   11   12   13   14   15   16
  17   18   19   20   21   22   23   24
  25   26   27   28   29   30
day0 day1 day2 day3 day4 day5 day6 day7 day8
                  1    2    3    4    5    6
   7    8    9   10   11   12   13   14   15
  16   17   18   19   20   21   22   23   24
  25   26   27   28   29   30
```

Problem S **Hole Cutter** 150 points

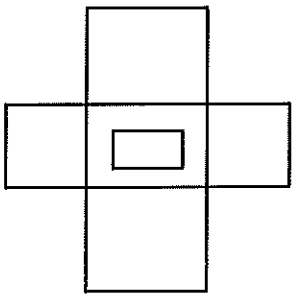
A factory which specialises in making cuts in the interior of flat sheets has just acquired a new cutter which can make cuts much more freely than any of their previous machines, and they want you to write a program to calculate exactly what has happened when a complex series of cuts are made. In particular, they need to know the number of holes which are formed in the sheet by the cuts. Here are some examples of situations that can arise after cutting:



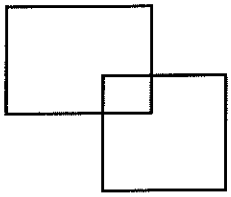
Two holes



Two holes



One hole



One hole

Input will be from standard input (ie the keyboard or a redirected file) and will consist of several cutting operation descriptions. Each description starts with a number, N, giving the number of cuts in the operation, followed by N lines giving the actual cuts. The number N will always be less than 100 and greater than 1. Each cut will be given by four whole numbers separated by one or more spaces, the first two giving the (x,y) coordinates of the start of the cut line and the second two defining the end of the cut line; the coordinate values will always be whole numbers less than 10000. You should assume the points are always internal to the sheet, never on the boundary. Each cut will be parallel to the x or y axis of the table. The input will be terminated by a line consisting of a single 0, ie a cutting operation description with N = 0.

Output, which must be written to standard output (the screen), must be one number for each cutting operation description in the input, giving the number of distinct holes formed by the cuts. Note that the minimum area of any hole is 1 square unit.

EXAMPLE

Input

```
4
0 1 1 1
1 1 1 0
1 0 0 0
0 0 0 1
2
0 1 2 1
1 2 1 0
0
```

Output

```
1
0
```

Problem T **Saturn** 150 points

You have come in the possession of a puzzle called the Saturn. It consists of two rings, holding 32 disks (16 per ring). Each disk has two (not necessarily different) colours, one colour a side. The disks are presented in the rings with one side (i.e. one of their colours) showing. Using an ingenious construction, the disks in each ring can be rearranged arbitrarily, a disk can be flipped over (therefore switching its colour), or any two disks can be swapped between rings.

The aim of the puzzle is to arrange and orientate the disks in such a way that each ring contains 4 colours of 4 disks each (giving a total of 8 different colours for the whole Saturn, each colour shown by 4 disks). Obviously, each disks can (and must) participate in only one colour.

After trying for some time, you still haven't made much progress. So, you decide to write a computer program which will solve problems of this type, and see if you can get it fast enough so that it will solve the main puzzle in under two minutes. The hard part is knowing which colour each disk should show; rearranging them is easy. Your solution must simply state the colour each disk is to show, in order to get an equal number of disks of each colour.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of a number of Saturn type data sets. Each set starts with a line containing two integers, giving the number of disks N and the number of different colours C that make up the solution respectively. This will be followed by N lines, each containing two integers representing the two colours for a disk. The file will be terminated with a pair of zeroes for the number of disks and colours. You may assume that N is a multiple of C. Note that the number of disks per colour in the solution will be N/C, but each colour in the input set may be present on any number of disks. There will be at most 32 disks.

Output consists of all the solutions for each data set. There must be one solution per line, and there must be a blank line between the solutions for each data set. A solution is a line containing information about which colour is shown on each of the disks, starting from the first disk in the input data. For each disk, write out the colour number shown on the disk. There must be one space between disks.

If there is no solution, write out the line "No solution". If there is more than one solution, the solutions must be ordered in the usual lexicographical order: the solution 1 1 2 2 comes before 1 2 1 2 which comes before 2 1 1 2. There should be no duplicate solutions in a set.

EXAMPLE:

Input

```
10 2
0 1
0 1
0 1
1 3
1 3
1 2
3 2
3 2
3 0
3 3
8 4
14 14
17 16
14 14
15 16
3 15
16 16
14 16
3 5
0 0
```

Output

```
1 1 1 1 3 1 3 3 3 3
1 1 1 3 1 1 3 3 3 3

No solution
```

Problem U **Coin Flipping** 150 points

Suppose a set of coins is arranged in a square, with some coins being head up, and others tail up:

```
H T T
H T T
T T T
```

Each coin may be flipped over, and when this happen, the adjacent coins turn too. Thus, if the top left coin is flipped, the arrangement becomes:

```
T H T
T T T
T T T
```

For this problem you have to find the fewest number of flips which will turn all the coins in the same direction, either heads or tails. For example, the strategy for the square shown is to flip the top right coin and the bottom centre coin, for a total of two flips, to get a square of all H.

Input will be from standard input (ie the keyboard or a redirected file) and will consist of data for several coin squares. Each data set starts with a line containing a single integer, giving N ($2 \le N \le 4$), the side length of the square the coins are arranged in. The file will be terminated by a line containing a single 0.

Output for each coin square, which must be written to standard output (the screen), must be a single number giving the minimum number of coin flips.

EXAMPLE

Input

```
3
HTT
HTT
TTT
2
HT
TT
4
THHH
HHHH
HHHH
HHHH
0
```

Output

```
2
2
6
```

Problem V **Foreign Exchange** **50-150 Points**

This problem involves the use of a visual programming language like Java. If your program performs according to the specifications given it will automatically earn 50 points. Up to 100 extra points can be earned by a well-designed interface the judges will look at the interface and make a decision (which will be available after the contest).

When money is transferred from the currency of one country to another, an official exchange rate is used to calculate the amount in the new currency. The quantity of money in the originating country is multiplied by this rate to give the quantity of money in the destination country. These exchange rates change several times during a day, independently in each country. Thus, if money is moved from country A to country B, and shortly afterwards from country B back to country A, there is a chance that the final amount will be more than the initial amount, although it will never be very much more (often it will be less). It is more likely that larger gains can be made when money is moved in a circle around several countries (for example, from NZ to Australia, Australia to Japan, Japan to the US, US to NZ), if the right country order is used.

For this problem, you must construct a screen which a money dealer can use to work out how to send money in order to maximise the gain. The route that the money takes will be described by a "country transfer list", giving the starting country, the first country the money is transferred to, and so forth back to the starting country again. Exchange rates for the country transfers will be entered on the screen, and a "total exchange rate" computed by multiplying these together. The money dealer needs to manipulate the country list until this "total exchange rate" comes out as much greater than 1 as possible.

Input will be from a file called Exchange.dat and the first line will give the country codes of the countries the dealer has contact with, all three-letter codes separated by space characters, for example:

NZL AUS JAP USA GBI OES DEU

There will be at most 10 countries. If there are N countries, then N lines will follow giving the exchange rates. Each exchange rate line will have N numbers on it giving the exchange rate from the "row" country to the "column" country. For example, with the country list above, the first exchange rate line will give the exchange rate when the $NZL is converted to the currency in the other countries, so it could be:

1.0000 0.8557 69.281 0.5061 0.3011 6.2814 0.9852

The second exchange rate line will give the exchange rate when the $AUS is converted to other currencies, so it could be:

1.1686 1.0000 80.964 0.5914 0.3519 7.3407 1.1513

Each exchange rate will be a decimal number with at least 1 digit in front of the decimal point and between 1 and 4 decimal places, and will always have 5 digits in it (so the largest possible exchange rate is 9999.9 and the smallest 0.0001). The "total exchange rate" must be rounded so it conforms to this 5-digit rule (the "total exchange rate" will always be within the limits given).

Your program must display the country codes on the screen, and allow the user to select up to 5 different country codes to form the country transfer list. When the user has indicated that the list is finished, your program will then display the relevant exchange rates and calculate, and display, the total exchange rate. The user must then be able to manipulate the country transfer list easily, by deleting countries, adding countries (up to the 5-country limit), changing the order of countries, and so forth. Your program must display exchange rates and the total exchange rate when the user requests a new calculation. The user must be able to stop the program when finished.

There is no output. Only the user interface is required.