



# XX Maratón Nacional de Programación 2006 ACM ICPC

## Problemas

(Este conjunto contiene 8 problemas; páginas numeradas de 1 a 16)



Regionals 2006  
**acm** International Collegiate  
Programming Contest



event  
sponsor

# Problem A

## Equidivisions

*Source file name: equidivisions.c, equidivisions.cpp or equidivisions.java*

An equidivision of an  $n \times n$  square array of cells is a partition of the  $n^2$  cells in the array in exactly  $n$  sets, each one with  $n$  contiguous cells. Two cells are contiguous when they have a common side.

A good equidivision is composed of contiguous regions. The figures show a good and a wrong equidivision for a  $5 \times 5$  square:

1	1	1	5	5	1	1	1	4	5
2	1	5	5	4	2	1	5	4	5
2	1	5	4	4	2	1	5	5	4
2	2	4	4	3	2	2	4	4	3
2	3	3	3	3	2	3	3	3	3

Note that in the second example the cells labeled with 4 describe three non-contiguous regions and cells labeled with 5 describe two non-contiguous regions. You must write a program that evaluates if an equidivision of the cells in a square array is good or not.

## Input

It is understood that a cell in an  $n \times n$  square array is denoted by a pair  $(i, j)$ , with  $1 \leq i, j \leq n$ . The input file contains several test cases. Each test case begins with a line indicating  $n$ ,  $0 < n < 100$ , the side of the square array to be partitioned. Next, there are  $n - 1$  lines, each one corresponding to one partition of the cells of the square, with some non-negative integer numbers. Consecutive integers in a line are separated with a single blank character. A line of the form

$$a_1 a_2 a_3 a_4 \dots$$

means that cells denoted with the pairs  $(a_1, a_2)$ ,  $(a_3, a_4)$ , ... belong to one of the areas in the partition. The last area in the partition is defined by those cells not mentioned in the  $n - 1$  given lines. If a case begins with  $n = 0$  it means that there are no more cases to analyze.

*The input must be read from the file equidivisions.in.*

## Output

For each test case **good** must be printed if the equidivision is good, in other case, **wrong** must be printed. The answers for the different cases must preserve the order of the input.

*The output must be written to standard output.*

Sample Input	Output for the sample input
2	wrong
1 2 2 1	good
5	wrong
1 1 1 2 1 3 3 2 2 2	
2 1 4 2 4 1 5 1 3 1	
4 5 5 2 5 3 5 5 5 4	
2 5 3 4 3 5 4 3 4 4	
5	
1 1 1 2 1 3 3 2 2 2	
2 1 3 1 4 1 5 1 4 2	
4 5 5 2 5 3 5 5 5 4	
2 4 1 4 3 5 4 3 4 4	
0	

## Problem B

### Generalized Matryoshkas

*Source file name: matriosh.c, matriosh.cpp or matriosh.java*

Vladimir worked for years making *matrioshkas*, those nesting dolls that certainly represent truly Russian craft. A matrioshka is a doll that may be opened in two halves, so that one finds another doll inside. Then this doll may be opened to find another one inside it. This can be repeated several times, till a final doll -that cannot be opened- is reached.

Recently, Vladimir realized that the idea of nesting dolls might be generalized to nesting toys. Indeed, he has designed toys that contain toys but in a more general sense. One of these toys may be opened in two halves and it may have more than one toy inside it. That is the new feature that Vladimir wants to introduce in his new line of toys.

Vladimir has developed a notation to describe how nesting toys should be constructed. A toy is represented with a positive integer, according to its size. More precisely: if when opening the toy represented by  $m$  we find the toys represented by  $n_1, n_2, \dots, n_r$ , it must be true that  $n_1 + n_2 + \dots + n_r < m$ . And if this is the case, we say that toy  $m$  *contains directly* the toys  $n_1, n_2, \dots, n_r$ . It should be clear that toys that may be contained in any of the toys  $n_1, n_2, \dots, n_r$  are not considered as directly contained in the toy  $m$ .

A *generalized matrioshka* is denoted with a non-empty sequence of non zero integers of the form:

$$a_1 \ a_2 \ \dots \ a_N$$

such that toy  $k$  is represented in the sequence with two integers  $-k$  and  $k$ , with the negative one occurring in the sequence first than the positive one.

For example, the sequence

$$-9 \ -7 \ -2 \ 2 \ -3 \ -2 \ -1 \ 1 \ 2 \ 3 \ 7 \ 9$$

represents a generalized matrioshka conformed by six toys, namely, 1, 2 (twice), 3, 7 and 9. Note that toy 7 contains directly toys 2 and 3. Note that the first copy of toy 2 occurs left from the second one and that the second copy contains directly a toy 1. It would be wrong to understand that the first  $-2$  and the last 2 should be paired.

On the other hand, the following sequences do not describe generalized matrioshkas:

•

$$-9 \ -7 \ -2 \ 2 \ -3 \ -1 \ -2 \ 2 \ 1 \ 3 \ 7 \ 9$$

because toy 2 is bigger than toy 1 and cannot be allocated inside it.

•

$$-9 \ -7 \ -2 \ 2 \ -3 \ -2 \ -1 \ 1 \ 2 \ 3 \ 7 \ -2 \ 2 \ 9$$

because 7 and 2 may not be allocated together inside 9.

•

-9 -7 -2 2 -3 -1 -2 3 2 1 7 9

because there is a nesting problem within toy 3.

Your problem is to write a program to help Vladimir telling good designs from bad ones.

## Input

The input file contains several test cases, each one of them in a separate line. Each test case is a sequence of non zero integers, each one with an absolute value less than  $10^7$ .

*The input must be read from the file matriosh.in.*

## Output

Output texts for each input case are presented in the same order that input is read.

For each test case the answer must be a line of the form

`:-) Matrioshka!`

if the design describes a generalized matrioshka. In other case, the answer should be of the form

`:-( Try again.`

*The output must be written to standard output.*

Sample input	Output for the sample input
-9 -7 -2 2 -3 -2 -1 1 2 3 7 9	<code>:-) Matrioshka!</code>
-9 -7 -2 2 -3 -1 -2 2 1 3 7 9	<code>:-( Try again.</code>
-9 -7 -2 2 -3 -1 -2 3 2 1 7 9	<code>:-( Try again.</code>
-100 -50 -6 6 50 100	<code>:-) Matrioshka!</code>
-100 -50 -6 6 45 100	<code>:-( Try again.</code>
-10 -5 -2 2 5 -4 -3 3 4 10	<code>:-) Matrioshka!</code>
-9 -5 -2 2 5 -4 -3 3 4 9	<code>:-( Try again.</code>

# Problem C

## Babylonian Roulette

*Source file name:* `broul.c`, `broul.cpp` or `broul.java`

... Los babilonios se entregaron al juego. El que no adquiría suertes era considerado un pusilánime, un apocado. Con el tiempo, ese desdén justificado se duplicó. Era despreciado el que no jugaba, pero también eran despreciados los perdedores que abonaban la multa ...  
Jorge Luis Borges, La lotería de Babilonia, Ficciones

People of Babylon were devoted to chance games and one of the most popular was a special kind of roulette. Recently, some old Babylonian tablets were found. They described details of the roulette game.

In modern terms, the rules of the game were as follows:

- Roulette's compartments had only six labels:  $-1$ ,  $-2$ ,  $-3$ ,  $1$ ,  $2$ ,  $3$ .
- The game was played by turns, during a day. Turns were numerated  $0$ ,  $1$ ,  $2$ ,  $\dots$
- Players could win or lose a multiple of the *bet*, a quantity of money that was constant along the day.
- At turn  $t$  there was an amount of money  $P_t$ , called the *pot*.
- At the start, there was an initial amount of money  $P_0$  in the pot.
- $P_0$  and the bet were positive numbers arbitrarily defined by the King.
- In a turn, a player turned the roulette. A player could not play more than once in a day. Depending on the compartment where the ball came to rest, the player won (or lose, if the value was negative) an amount  $w_t = L * bet$  of money, where  $L$  corresponded to the compartment's label.
- The won money was taken from the pot (or put in it if the player lose), i.e. the value of the pot in a given turn was determined by  $P_{t+1} = P_t + w_t$ .
- If as a result of the last rule  $P_{t+1}$  was a negative number the winner won only the maximum multiple of the bet that he could win without making a negative pot.
- If at some turn the pot was less than the bet, the game was ended for that day. If that was not the case the game continued till sunset.

Beside the tablets that explained the rules some other tablets were found. These had lines with three numbers. Archeologists conjecture that each of these lines were part of a kind of accountability system for the game, where numbers represented, for a given day, the value of the pot at the beginning, the bet and the value of the pot at the end.

For example, a line with the numbers

10000 1500 11500

could mean that there was only one turn where the player won with label 1. Another possibility is that there were three turns with results 2, 1 and -2.

On the other hand, there were found other tablets with triplets of numbers that seem like the above described that, however, cannot represent results of a game day. There is no hypothesis of what they are.

Archeologists want to validate their hypothesis analyzing batches of tablets with triplets. They want to estimate the number of people that played in a day. To begin, they want to establish, for each triplet of numbers in a tablet that could represent a result of a game day, the minimal number of players that played that day. In the above example the answer to this question is 1. Tablets that cannot represent results should be identified. You are hired to help with this task.

## Input

The input file contains several test cases, each one of them in a separate line. Each test case is a triplet of non negative integers, indicating the initial pot, the bet and the final pot for a day. Each of the input numbers is less than  $10^8$ . The initial pot and the bet are greater than 0.

A line with a triplet of 0's denotes the end of the input.

*The input must be read from the file broul.in.*

## Output

Output texts for each input case are presented in the same order that input is read. For each test case the answer must be a printed line.

If the test case cannot represent the result of a game day, the output line has the words **No accounting tablet**. In other case, the printed answer is one positive integer number telling the minimal number of players that could turn the roulette for the day corresponding to the annotations.

*The output must be written to standard output.*

Sample input	Output for the sample input
10000 1000 22000	4
24 13 2	No accounting tablet
5100 700 200	3
54 16 158	No accounting tablet
360 6 72	16
25 10 5	1
0 0 0	

# Problem D

## Continuous Fractions

*Source file name: cfrac.c, cfrac.cpp or cfrac.java*

A simple continuous fraction has the form:

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{\ddots + \frac{1}{a_n}}}}$$

where the  $a_i$ 's are integer numbers.

The previous continuous fraction could be noted as  $[a_1, a_2, \dots, a_n]$ . It is not difficult to show that any rational number  $\frac{p}{q}$ , with integers  $p > q > 0$ , can be represented in a unique way by a simple continuous fraction with  $n$  terms, such that  $\frac{p}{q} = [a_1, a_2, \dots, a_{n-1}, 1]$ , where  $n$  and the  $a_i$ 's are positive natural numbers.

Your task is to find and print the simple continuous fraction that corresponds to a given rational number.

### Input

Input will consist of a series of cases, each one in a line. A line describing a case contains  $p$  and  $q$ , two integer numbers separated by a space, with  $10^{20} > p > q > 0$ .

The end of the input is indicated by a line containing 0 0.

*The input must be read from the file cfrac.in.*

### Output

Cases must be analyzed in the order that are read from the input. Output for each case will consist of several lines. The first line indicates the case number, starting at 1, using the format:

**Case i:**

replacing  $i$  by the corresponding case number. The second line displays the input data in the form  $p / q$ .

The remaining lines must contain the continuous fraction corresponding to the rational number,  $\frac{p}{q}$ , specified in the given input line. The continuous fraction must be printed accordingly to the following rules:

- Horizontal bars are formed by sequences of dashes '-'.
  - The width of each horizontal bar is exactly equal to the width of the denominator under it.

- Blank characters should be printed using periods ‘.’
- The number on a fraction numerator must be printed center justified. That is, the number of spaces at either side must be same, if possible; in other case, one more space must be added at the right side.

*The output must be written to standard output.*

Sample input	Output for the sample input
<pre>75 34 65 60 0 0</pre>	<pre>Case 1: 75 / 34 .....1..... 2.+----- .....1.... ....4.+----- .....1.. .....1.+----- .....1 .....5.+.- .....1  Case 2: 65 / 60 .....1... 1.+----- .....1 ....11.+.- .....1</pre>

# Problem E

## Polygon Encoder

*Source file name: polygon.c, polygon.cpp or polygon.java*

Imagine an infinite table with rows and columns numbered using the natural numbers. The following figure shows a procedure to traverse such a table assigning a consecutive natural number to each table cell:

	0	1	2	3	4	5	6	7	8
0	0	2	5	9	14	20	27	35	.....
1	1	4	8	13	19	26	34	.....	
2	3	7	12	18	25	33	.....		
3	6	11	17	24	32	.....			
4	10	16	23	31	.....				
5	15	22	30	.....					
6	21	29	.....						
7	28	.....							
8	.....								

This enumeration of cells can be used to represent complex data types using natural numbers:

- A pair of natural numbers  $(i, j)$  is represented by the number corresponding to the cell in row  $i$  and column  $j$ . For instance, the pair  $(3, 2)$  is represented by the natural number 17; this fact is noted by  $P_2(3, 2) = 17$ .
- The pair representation can be used to represent  $n$ -tuples. A triplet  $(a, b, c)$  is represented by  $P_2(a, P_2(b, c))$ . A 4-tuple  $(a, b, c, d)$  is represented by  $P_2(a, P_2(b, P_2(c, d)))$ . This procedure can be generalized for an arbitrary  $n$ :

$$P_n(a_1, \dots, a_n) = P_2(a_1, P_{n-1}(a_2, \dots, a_n)),$$

where  $P_n$  denotes the  $n$ -tuple representation function,  $n \geq 2$ . For example  $P_3(2, 0, 1) = 12$ .

- A list of arbitrary length  $\langle a_1, \dots, a_n \rangle$  is represented by

$$L(\langle a_1, \dots, a_n \rangle) = P_2(n, P_n(a_1, \dots, a_n)).$$

For example,  $L(\langle 0, 1 \rangle) = 12$ .

The Association of Convex Makers (ACM) uses this clever enumeration scheme in a polygon representation system. The system can represent a polygon, defined by integer coordinates, using a natural number as follows: given a polygon defined by a vertex sequence  $\langle (x_1, y_1), \dots, (x_n, y_n) \rangle$  assign the natural number:

$$L(\langle P_2(x_1, y_1), \dots, P_2(x_n, y_n) \rangle).$$

ACM needs a program that, given a natural numbers that represents a polygon, calculates the area of the polygon. It is guaranteed that the given polygon is a simple one, i.e. its sides do not intersect.

As an example of the problem, the triangle with vertices at (1,1), (2,0) and (0,0) is codified with the number 2141. The area of this triangle is 1.

## Input

The input consists of several test cases. Each test case is given in a single line of the input by a natural number representing a polygon. The end of the test cases is indicated with \*.

*The input must be read from the file polygon.in.*

## Output

One line per test case, preserving the input order. Each output line contains a decimal number telling the area of the corresponding encoded polygon. Areas must be printed with 1 decimal place, truncating less significative decimal places.

*The output must be written to standard output.*

Sample input	Output for the sample input
2141	1.0
206	0.5
157895330	1.0
*	

# Problem F

## Uncle Jack

*Source file name: uj.c, uj.cpp or uj.java*

Dear Uncle Jack is willing to give away some of his collectable CDs to his nephews. Among the titles you can find very rare albums of Hard Rock, Classical Music, Reggae and much more; each title is considered to be unique. Last week he was listening to one of his favorite songs, *Nobody's fool*, and realized that it would be prudent to be aware of the many ways he can give away the CDs among some of his nephews.

So far he has not made up his mind about the total amount of CDs and the number of nephews. Indeed, a given nephew may receive no CDs at all.

Please help dear Uncle Jack, given the total number of CDs and the number of nephews, to calculate the number of different ways to distribute the CDs among the nephews.

### Input

The input consists of several test cases. Each test case is given in a single line of the input by, space separated, integers  $N$  ( $1 \leq N \leq 10$ ) and  $D$  ( $0 \leq D \leq 25$ ), corresponding to the number of nephews and the number of CDs respectively. The end of the test cases is indicated with  $N = D = 0$ .

*The input must be read from the file uj.in.*

### Output

The output consists of several lines, one per test case, following the order given by the input. Each line has the number of all possible ways to distribute  $D$  CDs among  $N$  nephews.

*The output must be written to standard output.*

Sample input	Output for the sample input
1 20	1
3 10	59049
0 0	

# Problem G

## Babel Towers

Source file name: `babel.c`, `babel.cpp` or `babel.java`

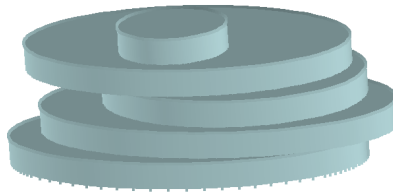
Babel Inc. is a company that designs structures for skyscrapers. They have developed a simple technique to estimate the feasibility of a design, based on the fact that their designs are equivalent to stack blocks of circular section and constant height. The height of a block is used as the unit of lineal measure.

*Blocks* are all made of the same non deformable uniform material, so that the weight of a block is proportional to its volume and its center of mass is the geometrical center of the block. A typical block is of the form



which weight is determined by the radius of the circular section (because every one has height 1).

For  $n \geq 1$ , an  $n$ -tower is a stack of  $n$  blocks. Blocks in an  $n$ -tower are numbered  $0, 1, \dots, n-1$ , from the bottom to the top. Given an  $n$ -tower, its  $k$ -subtower is defined as the  $k$ -tower with the first  $k$  blocks of the  $n$ -tower,  $0 \leq k < n$ . An example of a 5-tower could look like the following figure:



A tower is *feasible* when it can be built putting its blocks one by one from bottom to top. At each building step the corresponding subtower should be *stable*, i.e., every block of it must remain in the position that it is designed to stay. When placing a block results in a stack of blocks which part of it has a center of mass that lies out of the base on that it is supported we say that the resulting tower *collapses* and is *unfeasible*.

Babel Inc. does not consider stable a situation where the center of mass of the system of blocks that are supposed to rest on a block lies on the border of the supporting surface. For instance, two blocks of the same dimensions with one of them placed centered on the circumference of the other constitute a 2-tower that collapses; but if the second one is placed within the circle of the first one the system is feasible.

Your problem is to help Babel Inc. in the evaluation of tower designs. Given a design for a  $n$ -tower you must judge if it is feasible and, if it is not, you must tell at which floor it would collapse. In this last case you give  $k$  as answer, with  $0 < k < n$ , if the  $(k-1)$ -subtower is feasible but the  $k$ -subtower collapses.

## Input

The input file contains several test cases.

Each test case begins with a line indicating  $N$ ,  $0 < N < 1000$ , the number of blocks that the designed tower should have.

Then, there is a line describing each one of the blocks. The block  $k$ ,  $0 \leq k < N$ , is described by a line containing 3 integers separated by blanks, of the form

$$x_k \ y_k \ r_k$$

which indicates that the block  $k$  with radius  $r_k$  must be placed supported on the block  $k - 1$  (exception: block 0 is placed on the ground) with its center of mass on the point  $\langle x_k, y_k, k + 0.5 \rangle$  with respect to a grid with origin in  $\langle 0, 0, 0 \rangle$ , for  $x_k, y_k \leq |10^5|$  and  $1 \leq r_k \leq 10^5$ .

A case begin with  $N = 0$  means that there are no more cases to analyze.

*The input must be read from the file `babel.in`.*

## Output

Output texts for each input case are presented in the same order that input is read.

For each test case the answer must be of the form ‘Feasible’ if the design is feasible. In other case, the answer should be of the form ‘Unfeasible  $k$ ’ indicating that the  $(k - 1)$ -subtower is feasible but the  $k$ -subtower is unfeasible.

*The output must be written to standard output.*

Sample input	Output for the sample input
3	Feasible
0 0 10	Unfeasible 3
2 0 12	Feasible
-4 1 1	Unfeasible 1
4	
0 0 12	
0 0 10	
0 9 10	
0 17 5	
4	
0 0 4	
0 1 4	
1 0 4	
-1 -1 4	
2	
10 10 5	
0 0 3	
0	

# Problem H

## Little Quilt

*Source file name: quilt.c, quilt.cpp or quilt.java*

*Little Quilt* is a small language introduced by Ravi Sethi in his book ‘Programming Languages’. Here, a restricted version of Little Quilt is presented.

The language is defined by the following BNF grammar:

$$\langle \text{QUILT} \rangle ::= A \mid B \mid \text{turn}(\langle \text{QUILT} \rangle) \mid \text{sew}(\langle \text{QUILT} \rangle, \langle \text{QUILT} \rangle)$$

A and B represent the two primitive quilts. Each primitive quilt corresponds to a matricial arrangement of  $2 \times 2$  characters. `turn()` and `sew()` are operations over quilts.

The instruction `turn(x)` turns the quilt `x` 90 degrees clockwise. The following table illustrates the primitive quilts as well as examples of the effect of the `turn()` operation:

A	// /+
turn(A)	\\ +\
turn(turn(A))	+/ //
turn(turn(turn(A)))	\+ \\
B	-- --
turn(B)	 

Accordingly, the instruction `sew(x,y)` sews quilt `x` to the left of quilt `y`. Both `x` and `y` must have the same height, otherwise an error will be generated. The following figure represents the result of `sew(A,turn(B))`:

```
//||  
/++||
```

while the `sew(turn(sew(B,turn(B))),A)` generates an error message.

Your job is to build an interpreter of the Little Quilt language.

## Input

The input file will be a text file containing different Little Quilt expressions, each one ended by a semicolon character (;). Space and new line characters must be ignored; this means that an expression may span several lines.

*The input must be read from the file quilt.in.*

## Output

The output file contains the quilts produced as a result of interpreting the input expressions.

Each quilt must be preceded by a line, left aligned, with the format

**Quilt i:**

where *i* is the quilt number, starting at 1. If the expression interpretation generates an error, the word

**error**

must be printed.

*The output must be written to standard output.*

Sample input	Output for the sample input
<pre> sew(turn(sew(B,turn(B))),       turn(sew(turn(B),B)))  ;        sew(turn(sew(B,turn(B))),A); sew(turn(sew(A,turn(A))),       turn(turn(        turn(sew(A,turn(A))))))  ; </pre>	<pre> Quilt 1:   --   -- --   --   Quilt 2: error Quilt 3: \\// +\\/+ +\\/+ /\\/ </pre>