ACM Asia Collegiate Programming Contest 1998

Notes for Contestants

1. There are 8 problems for the contest. You may work on the problems in any order.

2. You may use any algorithm/method to solve the problems. However, the execution time of your program for each problem must not exceed 30 seconds, otherwise the program will be considered **run-time exceeded**.

3. The judging system, **PC^2**, is provided by ACM. For detailed information of system environment and operational instructions, please refer to the additional *Contest handbook*.

4. Input files for the problems are preinstalled in the system for automatic judging. The input files are named according to the problem ID, where "**px.in**" is the input file for **Problem X**.

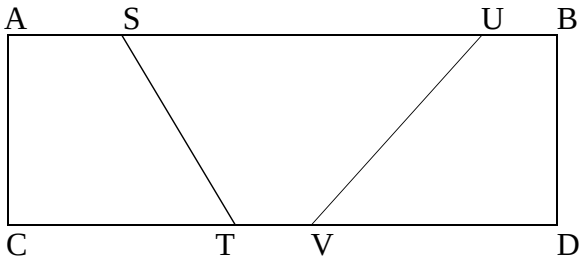
5. If you have any questions during the contest, you may communicate with the judges by sending message to the judges through the *Clarification System* in **PC^2**.

Problem A

Who Gets the Largest Piece

Input file: pa.in

In the rectangle ABCD below, line segments ST and UV separate the rectangle into three sections. One can think of ABCD as a cake and use a knife to cut it along the lines ST and UV. Here S, T, U, and V must be on the perimeter of ABCD. To know which piece of cake is the largest, one can compute the area for each of these three sections. You are to write a program to calculate the sizes of various shapes after two cuts separate ABCD. Display these three or four sizes in a descending order, i.e., the area of the largest section followed by the nest largest, and so on. Each size must be printed exact to two digits to the right of the decimal point.Use your program to process *N* rectangles and select the largest section you find.



Input

The first line of the input file contains a positive integer *N* indicating the number of data sets to follow. Each of the following *N* lines contains the *x* and *y* coordinates of points A, B, C, D, S, T, U, and V respectively. These sixteen integers are separated by spaces.

Output

For each data set in the input file, output the areas of the separated sections in a descending order. Use spaces to separate these values. At the end, output the largest area from the *N* data sets in the last line.

Sample Input

```
2
0 4 16 4 0 0 16 0 4 4 8 0 15 4 10 0
-1 5 5 5 -1 0 5 0 0 5 0 0 -1 3 5 3
```

Output for the Sample Input

```
26.00 24.00 14.00
15.00 10.00 3.00 2.00
26.00
```

Problem B

Road Map

Input file: pb.in

A road map is constructed from numbers of piecewise intervals. Each piecewise interval is represented by two end points encompassed with parenthesis, e.g. ((100, 250), (250,350)). Each point belongs to a point in the 2D plane with two integer axis values, i.e., *x* value and *y* value. Define the terminology FIRST SEE set means that, given a specific point, all the piecewise intervals which can be seen at the point. In other meaning, the FIRST SEE set defines the set of piecewise intervals which are not completely blocked by other intervals with the standpoint of the point in the 360° directions. Given a series of piecewise intervals represents some specific city road map and a given specific point, write a program to find the FIRST SEE set of the point.

Input

The input file contains a series of piecewise intervals which represent some specific road map and finally a given specific point. Each line specifies the interval, the given point and ends with *.

Line 1: ((100,250),(250,350))
Line 2: ((u,v),(s,t))
...
Line n: (p1,p2)
Line n+1: *

Output

The output contains several lines. Each line represents an element of the FIRST SEE set of the given specific point (the set should be sorted in a increasing order with the sum of axis x value and axis y value of the first point of the intervals firstly, and the sum of the second point of the intervals secondly).

Sample Input

```
((120,150),(180,300))
((150,450),(230,450))
((180,550),(200,460))
((100,100),(200,110))
((500,380),(550,250))
((100,100),(150,130))
(200,200)
*
```

Output for the Sample Input

```
((100,100),(150,130))
((100,100),(200,110))
((120,150),(180,300))
((150,450),(230,450))
((500,380),(550,250))
```

Problem C

Test Compression

Input file: pc.in

In this assignment we consider the problem of compressing a *text string* using a *dictionar* . The dictionary consists of (*dictionary string*, *code word*)-pairs, each representing the code that can be used to replace a substring in the text. The code words are binary bit strings, not necessarily of equal length. The *text compression problem* is that of, given a *text string* over some finite alphabet and a *dictionar* , deciding the length of the shortest binary bit string which encodes the original text string.

Example 1.

text string: **abcdef**
Dictionary:

Dictionary String	**a**	**abc**	**abcd**	**bcd**	**def**	**ef**
Code word	**01**	**0**	**1011**	**1**	**10**	**11**

What follow are three different ways of encoding the text string:

Method 1

a	**bcd**	**ef**
01	**1**	**11**

Method 2

abc	**def**
0	**10**

Method 3

abcd	**ef**
1011	**11**

Clearly, Method 2 results in the shortest compressed string whose length is 3. Your job in this assignment is to find out the length of the shortest compressed string. If the text string cannot be compressed using the given dictionary, **0** should be returned. (For example, the text string **abcxxx** cannot be compressed using the dictionary in Example 1.)

The first line of the input file contains the number of text compression problems. The first line of each text compression problem contains the text string to be compressed. Following this are (**dictionary string, code word**)-pairs, each of which occupies a line of the input file. (For the sake of simplicity, we assume that the text string is over the alphabet which consists of the 26 lower-case characters '**a, b, c, ..., z**'. Examples of such text strings include: **abcdef**, **thisisatextstring**, and **acmprogrammingcontest**.

For each text compression problem, output the length of the shortest compressed string. Output one answer per line.

With respect Example 1, the input and output files are the following.

Sample Input

```
2
abcdef
(a, 01)
(abc, 0)
(abcd, 1011)
(bcd, 1)
(def, 10)
(ef, 11)
aa
(a,1)
(ab,10)
```

Output for the Sample Input

```
3
2
```

Problem D

Gambling

Input file: pd.in

A cheating casino creates an electronic symbol-bar machine. This machine has 3 belts attached to 3 wheels. Symbols from A to ◎ , are painted on each belt in alphabetic order. Figure 1 shows the display of this kind of machine. Totally there are 113 symbols printed on each belt. These symbols are orderly listed in Table 1. A belt will display the same symbols on the screen if a wheel rolls 113 steps. When the bar handle is pulled, each wheel will roll certain steps, according to three formulas. For example, if (6* *N*+3) is the formula for the first wheel's step-movement, where *N* is the N^{th} times the bar handle is pulled after the machine is reset, the first wheel will roll (6**N*+3) steps and show another 3 symbols on the screen. Assume each wheel rolls on the same direction, from **bottom to top** according to Figure 1. Each wheel has different formula for directing how many steps to move its belt. The three formulas are specified as input data in this problem. Once 3 ☺s are displayed on the center row of the screen, it means a BINGO. On hundred times the bet will be rewarded to the gambler.

L	L	L
M	M	M
N	N	N

Figure 1: The display of the symbol-bar machine

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z	α	β	χ	δ	ε	ϕ	γ	η	ι	φ	κ	λ	μ	ν
π	θ	ρ	σ	τ	υ	ϖ	ξ	♋	♌	♍	♎	♏	♐	♑	♒	♓	🙰	🙱	●
■	□	⛆	◆	❖	☒	⌘	✌	🤞	☜	☞	☝	☹	☺	😐	🖗	💣	🏳	✈	☼
,	.	/	ə	;	7	4	3	Δ	℃	⊕	ㄅ	ㄆ	ㄇ	ㄈ	ㄉ	ㄊ	ㄋ	ㄌ	ㄍㄍ
ㄎ	ㄏ	ㄐ	ㄑ	ㄓ	ㄔ	ㄕ	ㄖ	ㄠ	ㄝ	ㄞ	ㄤ	◎							

Table 1: The symbols printed on the belts

Now, a gambler enters into this casino and finds one *Symbol*-bar machine, which shows three symbols on the central line of the display. These three symbols are also specified as input data in this problem. This gambler wants to know how many times that he needs to pull the bar-handle to win a BINGO prize. Please write a program to help him. Further constraints and assumptions to this problem are specified as follows.

1. The initial symbols displayed on the center row of the screen after the machine is reset are specified in the input file, too. For example, 12, 58 113 means the first 3 symbols showed on the center row of the screen are L, ☙, and ◎ , after the machine is reset. The first value is used to show the initial symbol showed on the screen of the left wheel, while the second and third value are used to specify the symbols on the middle and right wheels.

2. Following above input data are three formulas. The formulas are all in the ($a * N + b$) format, where N is the Nth time the bar handle is pulled after the machine is reset. The values of a and b are used as the input data. There are 6 values specified in the input file. The first two numbers are used for controlling the left wheel. The second two numbers are used for controlling the middle wheel, and the last two numbers are used for controlling the right wheel.

3. Another following input data are 3 values used to specify the symbols showed in the screen at the time the gambler chooses that machine. This means after the machine was reset, there are people before the gambler chosen this machine and pulled the machine bar in certain times, which causes the machine to roll certain steps to show the current symbols on the screen. We assume it is the first time, after the machine was powered on, this pattern displayed on the screen when the gambler chooses this machine.

4. The output data is a number to indicate how many times the bar-handle needs to be pulled by the gambler to win a BINGO.

5. Instead of a number to show the number of times the bar-handle is pulled, if it is impossible for this gambler to win a BINGO, print “impossible”.

6. Each line of the input file is a configuration, which is nothing related to the other lines. A ‘*’ in the last line of the input file represents the end of the input data.

Input

The input file contains (K+1) lines. Each line specifies the input data for each test.

Line 1: U, V, W, a, b, c, d, e, f, A, B, C
Line 2: P, Q, R, s, t, u, v, w, x, D, E, F
...
Line K: I, J, K, m, n, o, p, q, r, X, Y, Z
Line (K+1): * ⇒ end of input file

Output

The output file contains K lines. Each line is the output for each input configuration listed in input file.

Line 1: abcd ⇒ a numerical number
Line 2: efgh ⇒ a numerical number
...
Line K: wxyz ⇒ a numerical number

Sample Input

```
74 74 74 0 1 0 1 0 1 75 75 75
1 1 1 0 1 0 1 0 1 77 77 77
3 5 7 0 1 0 1 0 1 75 75 75
*
```

Output for the Sample Input

```
112
110
impossible
```

Problem E

Expression Evaluation

Input file: pe.in

Write a program that can understand and calculate the infix notation expressions. All the values given and to be evaluated are considered to be only *integers*. The possible symbols of the expression include *integer values, arithmetic operators*, *special symbols, keywords,* and *variables*.

The values are just ordinary integral literal strings. The arithmetic operators are ordinary binary operators including '+', '-', '*', and '/'; the special symbols include the equal sign '=', parentheses, '(' and ')', and the question mark '?'. The possible variables are just single English alphabets ranging from 'a' to 'z'; these variables are initialized as zeros at first.

Each input line contains an assignment expression with a variable on the left-hand side, following by the equal sign in the middle. The right-hand side of the assignment expression will be an ordinary integral expression (possible contains several variables); the value of the expression shall be assigned to the variable for later evaluation. If the right hand side expression is a question mark, output the value of the left-hand side variable. Further, the possible keywords in the input file include '**if**', '**while**', and '**end**' statements. The syntax and meaning of while statement is as follows:

```
while ( [exp] )
      [assignment-statements]
end
```

Here the interpreter will evaluate the value of [exp], if the value of [exp] is not zero, then the [assignment-statements] will be repetitively evaluated until the value of [exp] equals to zero. The syntax and meaning of **if** statement is just like the **while**-statement except that the [assignment-statements] will be evaluated just once if the value of [exp] is not zero.

Note:

(1). The division operator is supposed to be the *integer division*. For example, 13 / 4 shall be just 3, instead 3.25.

(2). For simplicity, no nested statement is allowed with '**if**' and '**while**' statements. That is, the only possible statements within these blocks are assignment statements.

Input

Several lines of expression described above. A single '0' (zero) in the input line signifies the end of input.

Output

For each question marked expression, output the corresponding value of the variable in a line. Output a single star '*' to signify the end of outputs.

Sample Input

```
x = 4
y = 3 * x - 2
y = ?
z = 5 + 2 * y
z = ?
x = 7 - y / 3
x = ?
while ( x )
   z = 2 * z
   x = x - 1
end
z = ?
0
```

Output for the Sample Input

```
10
25
4
400
*
```

Problem F

Prime Polynomials

Input file: pf.in

Let $(F, +, \bullet)$ be a field, and $F[x]$ be the set of all polynomials in x. That is, a polynomial $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 x^0$ is in $F[x]$ if and only if $a_i \subset F$ for every $0 \le i \le n$. If $a_n \neq 0$, then a_n is called the leading coefficient of $f(x)$, and we say that $f(x)$ has degree n.

Let the field be Z_2, there are only two elements, 0 and 1, in Z_2. The addition in Z_2 is defined as: $0 + 0 = 0, 0 + 1 = 1 + 0 = 1, 1 + 1 = 0$. The subtraction operation is the same as addition. That is $0 - 0 = 0, 0 - 1 = 1 - 0 = 1, 1 - 1 = 0$. The multiplication in Z_2 is defined as: $0 \bullet 0 = 0, 0 \bullet 1 = 1 \bullet 0 = 0, 1 \bullet 1 = 1$.

Let $f(x)$ and $g(x)$ be two polynomials in $Z_2[x]$. The addition, subtraction, multiplication, and division of $f(x)$ and $g(x)$ are defined similar to its corresponding operation of the ordinary polynomials, except that all operations should be computed in Z_2 as defined above. For example, $(x^2 + x^1) \bullet (x^1 + 1) = x^3 + x^2 + x^2 + x^1 = x^3 + x^1$.

A polynomial $f(x)$ is *prime* in $F[x]$, if there is no polynomials $g(x)$ and $h(x)$ in $F[x]$ with degree at least 1, satisfying $f(x) = g(x) \bullet h(x)$. In this problem, you are going to rite program to find prime polynomials of degree 1 to degree n, where n is a positive integer.

Input

The input file contains only one positive integer n. You may assume that $n < 23$.

Output

The output file should contain a set of prime polynomials in $Z_2[x]$. For each degree k, $1 \le k \le n$, list the prime polynomial $a_k x^k + a_{k-1} x^{k-1} + \ldots + a_0 x^0$ with smallest value of $a_k 2^k + a_{k-1} 2^{k-1} + \ldots + a_0 2^0$. The format of the output file is shown in the Output for the Sample Input section.

Sample Input

```
5
```

Output for the Sample Input

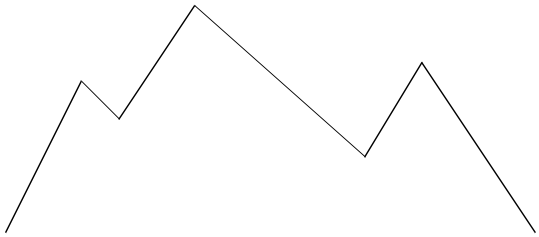
```
x^1
x^2+x^1+x^0
x^3+x^1+x^0
x^4+x^1+x^0
x^5+x^2+x^0
```

Problem G

Mountain Cribbing

Input file: pg.in

The following figure shows an example of a two-dimensional mountain range.



Two people start at the same elevation on the opposite sides of a two-dimensional mountain range. They want to move to the summit of the mountain range in such a way that they are at the same altitude every moment. Write a program to find a way for the two people to reach the summit of the mountain range in minimum steps.

Input

The input file may contain more than one two-dimensional mountain range. Each one begins with a number n, and then followed by n+1 coordinates. Each coordinate consists of a pair of integers. Each of the coordinates represents a peak or a valley of the mountain range. The last mountain range is followed by a 0, indicating the end of the input file. You may assume that n<30.

If we denote i-th coordinate of a two-dimensional mountain range as (x_i, y_i), $i = 1, ..., n+1$, then $x_1 < x_2 < ... < x_{n+1}$, $y_1 = y_{n+1} < \min(y_2, y_3, ..., y_n)$, and there exists a peak (x_j, y_j), $1 < j < n+1$, such that $y_j > \max(y_1, ..., y_{j-1}, y_{j+1}, ..., y_{n+1})$.

Output

For each two-dimensional mountain range, print a way for the two people to reach the summit of the mountain range in minimum steps. The output of each solution consists of a sequence of steps. Each step has two coordinates. The first one represents the location of the first person, and the second one represents the location of the second person. All the numbers should be printed exact to two digits to the right of decimal point. Print one step per line. Print a blank line after each solution.

Sample Input

```
6
0 0
2 2
3 1
7 5
11 1
13 3
16 0
2
0 0
1 1
2 0
0
```

Output for the Sample Input

```
(0.00,0.00) (16.00,0.00)
(2.00,2.00) (14.00,2.00)
(3.00,1.00) (15.00,1.00)
(5.00,3.00) (13.00,3.00)
(3.00,1.00) (11.00,1.00)
(7.00,5.00) (7.00,5.00)

(0.00,0.00) (2.00,0.00)
(1.00,1.00) (1.00,1.00)
```

Problem H

Move Grid Puzzle

Input file: ph.in

The following figure shows a 3×3 grid puzzle.

0	5	8
7	4	3
2	6	1

Formation of the grid puzzle:

1. The dimension is always 3×3.
2. ‘0’ is always in left top corner. ‘1’ is always in right bottom corner.
3. The rest grid are numbers between 2 ~ 8.

Movement:

‘0’ stands for blank cell. You may move the number next to ‘0’ to blank cell.
Then ‘0’ will occupy the cell that the number moves away.
The cost of this movement is exactly the number moved.

For example:

0	5	8
7	4	3
2	6	1

Move ‘5’ to ‘0’ and cost 5.
Or you may move ‘7’ to ‘0’ and cost 7.
But you can **not** move ‘4’ to ‘0’.

Goal of this problem:

Move ‘1’ to left top corner of the grid. And accumulated all the cost of movements.
There will be many ways to accomplish this task.
But the accumulated cost will be different.
Find the minimum accumulated cost and then print it out.

For example:

Original data:

0	5	8
7	4	3
2	6	1

Step 1: cost: 5

5	0	8
7	4	3
2	6	1

Step 2: cost: 8

5	8	0
7	4	3
2	6	1

Step 3: cost: 3

5	8	3
7	4	0
2	6	1

Step 4: cost: 1

5	8	3
7	4	1
2	6	0

.
.
.

Step N-1: cost: ?

0	1	X
X	X	X
X	X	X

Step N: cost: 1

1	0	X
X	X	X
X	X	X

Accumulated cost = 5 + 8 + 3 + 1 + ... + ? + 1

Input

The first line of input contains number of puzzles to solve. Each puzzle contains 9 numbers, 3 in a row and 3 rows. The first number of each puzzle must be 0, and the last one is 1.

Output

For each puzzle, print out the minimum accumulated cost.

Sample Input

```
2
0 5 8
7 4 3
2 6 1
0 5 3
8 2 4
6 7 1
```

Output for the Sample Input

```
46
44
```