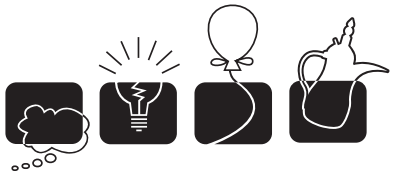


The 29th Annual ACM International Collegiate Programming Contest

Sponsored by IBM

Arab and North African Seventh Regional Contest

sponsored by:
Kuwait University
Kuwait Foundation for the Advancement of Science
NCS, IBM, & Projacs

Kuwait University
December 2, 2004

The problem set is made of 12 numbered pages

[A] Da Vinci's Cryptex

Program:	`cryptex.(c|cpp|java)`
Input:	`cryptex.in`
Balloon Color:	`Yellow`

Description

Leonardo Da Vinci, the famous inventor and artist, was fond of cryptography and has invented many devices and techniques to hide messages. One such invention is the cryptex. A *cryptex* is a small device used to carry a secret message and is made of one or more rings. Each ring has the 26 uppercase letters written in some random order. It is by aligning these rings in one specific way that the secret message is revealed. The secret message is made of two words, each of length N. The first word of the secret message is called the *unlocking word* and the second is called the *secret word*. To properly align the cryptex, you need to know the unlocking word. Once you have the cryptex and the unlocking word, all you have to do is align the rings on the cryptex to spell the unlocking word. The letters on the rings, though randomly ordered, are arranged in such a way that when the cryptex is aligned to spell the unlocking word, one of the other 25 strings would spell the secret word. To reveal the secret message you'll need to know at least one letter from the secret word.

Take for example the following cryptex made of five rings (each line constitutes a ring:)

```
KFZLQMDWJUSHGCEIXRAOPNVTYB
IMWZPFJBKLTNOEQDHUXGVYASRC
FAMIETZORWPSQUNGLDYBKXHCVJ
XNAKVPICQHDFWEGBRTMLZOUSYJ
ZSYFDOWIJCAKPBTXLRUNGQMVHE
```

The unlocking word is *"GREEN"* and we know that the second letter of the secret word is *"P"*. By aligning the rings to spell the unlocking word, the cryptex now looks like this:

```
GCEIXRAOPNVTYBKFZLQMDWJUSH
RCIMWZPFJBKLTNOEQDHUXGVYAS
ETZORWPSQUNGLDYBKXHCVJFAMI
EGBRTMLZOUSYJXNAKVPICQHDFW
NGQMVHEZSYFDOWIJCAKPBTXLRU
```

The secret word is revealed by looking for the word whose second letter is 'P'.

Input Format

Your program will be tested on one or more test cases. The first line of the input specifies a single integer D which represents the number of test cases. Each test case is specified using $N+1$ lines. The first line of each test case has the following format:

```
N U S
```

N is a positive integer which is the number of rings. No cryptex will have more than 1000 rings. U is the unlocking word while S describes the secret word. S is made of N characters, all underscore characters (`'_'`) except exactly one. For example, having S equal to `"_P___"` says that the second letter of the secret word is `"P"`.

The remaining N lines specifies the N rings, one on each line. Each line is made of (different) 26 uppercase letters.

Consecutive test cases are separated by a single blank line.

Output Format

For each test case, write on a separate line, the unlocking word, followed by the secret word, separated by a single space.

Sample Input/Output

cryptex.in

```
2
5 GREEN _P___
KFZLQMDWJUSHGCEIXRAOPNVTYB
IMWZPFJBKLTNOEQDHUXGVYASRC
FAMIETZORWPSQUNGLDYBKXHCVJ
XNAKVPICQHDFWEGBRTMLZOUSYJ
ZSYFDOWIJCAKPBTXLRUNGQMVHE

3 YES S__
POGCSAVYFENXBLUWTDRHJKZMIQ
DQVPBIJFEHNAGKMLXOCRTSZUWY
OURICSLEAMNQFDPYVHXGJWTZBK
```

OUTPUT

```
GREEN APPLE
YES SIR
```

[B] F of a and b to the k

Program:	`fun.(c|cpp|java)`
Input:	`fun.in`
Balloon Color:	`Blue`

Description

Consider the following function:

$$f^k(a,b) = \begin{cases} a & k = 0 \\ b & k = 1 \\ f^{k-1}(a,b) + f^{k-2}(a,b) & k > 1 \end{cases}$$

Given a pair of values **<*V*,*K*>**, write a program to determine if there are values for $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{k}$ satisfying all of the following:

$$0 < a \leq b < 10$$
$$0 \leq k \leq K$$
$$f^k(a,b) = V$$

Input Format

Your program will be given a series of pairs **<*V*,*K*>**, each pair on a separate line. Note that $0 \leq \boldsymbol{V} < 1,000,000$.

The end of the input file is identified by a dummy pair (which is not part of the test cases) where both $\boldsymbol{V}$ and $\boldsymbol{K}$ are 0.

Output Format

For each given pair **<*V*,*K*>**, if there are values for $\boldsymbol{a}$, $\boldsymbol{b}$,and $\boldsymbol{k}$ satisfying the requirements, write on a separate line, the following output:

```
f^k(a,b)=V
```

If there is more than one solution, print the solution with the smallest value of $\boldsymbol{k}$ within all the solutions with the smallest value of $\boldsymbol{b}$ within all the solutions with the smallest value of $\boldsymbol{a}$. (In other words, the first solution if they were sorted lexicographically in an ascending order using the key $<\boldsymbol{a},\boldsymbol{b},\boldsymbol{k}>$.)

If there is no solution, print the following:

```
no␣solution␣for␣<V,K>
```

Sample Input/Output

fun.in

```
5 10
5 1
34 3
34 4
34 20
0 0
```

OUTPUT

```
f^4(1,1)=5
f^1(1,5)=5
no solution for <34,3>
f^4(5,8)=34
f^8(1,1)=34
```

[C] The Long Divide

Program:	`divide.(c|cpp|java)`
Input:	`divide.in`
Balloon Color:	Green

Description

```
    5  r  2
4 | 22
    20
     2
```

Do you remember how hard it was when you were first taught how to perform long division? Until today, so many students (and grownups, for that matter,) find it difficult to perform a long division by hand. In case you forgot, take a look at the simple problem on the right, 22 divided by 4 is 5 and the remainder is 2. In this example, 22 is the dividend, 4 is the divisor, 5 is the quotient, while 2 is the remainder.

Write a program that illustrates all the steps performed in a long division. Note: In this problem, it is important that the output of your program matches the output of the judge's program in every detail. Read the instructions given in the output format, and study the given examples.

Input Format

Your program will be tested on several test cases. Each test case is specified on a separate line. For each test case, two integers will be given: first the dividend, then the divisor. Both are positive numbers less than $1,000,000,000$. The end of the test cases is a line whose divisor is 0 (which, of course, is not part of the test cases.)

Output Format

Your output must adhere to the following points:

1. The output of the test cases will be *separated* by a single blank line. Otherwise, the output doesn't contain any blank lines.
2. There is exactly one space character to the left of the divisor.
3. There should be no trailing spaces in any of the lines.
4. There should be no leading zeros in any of the numbers.
5. There should be no unnecessary multiplication and/or subtraction steps, as, for example, resulting from multiplying by 0.
6. Digits, spaces, and separation bars (made of one or more '-') should be of exact length and alignment as in the examples.

Sample Input

`divide.in`

```
22 4
102478 83
3811 37
10 83
0 0
```

Sample Output *(the gray vertical lines are for illustration purpose only. Use them to determine the location of spaces and alignment.)*

OUTPUT

```
     5 r 2
   +---
4  | 22
     20
     --
      2


        1234 r 56
    +-------
83  | 102478
      83
      --
      19478
      166
      ---
       2878
       249
       ---
        388
        332
        ---
         56

      103 r 0
    +-----
37  | 3811
      37
      --
       111
       111
       ---
         0

      0 r 10
    +---
83  | 10
```

[D] Partial Overlapping

Program:	overlap.(c|cpp|java)
Input:	overlap.in
Balloon Color:	Red

Description

Imagine a rectangular grid of letters of size $R \times C$. As you can see in the adjacent figure, two exact copies of this grid may be *partially* overlapped just by sliding one of the copies horizontally and/or vertically.

Write a program that prints the resulting shape of such overlapping grids.

A	B	C	D	E	F
G	H	A	B	C	D
I	J	G	H	A	B
K	L	I	J	G	H

A	B	C	D	E	F		
G	H	A	B	C	D	E	F
I	J	G	H	A	B	C	D
K	L	I	J	G	H	A	B
		K	L	I	J	G	H

Input Format

Your program will be tested on one or more test cases. Each test case is described using $R+1$ lines. The first line contains two positive integers: R is the grid's number of rows and C is the number of columns. The data in the grid is specified on the following R lines. each line represents a row and is made of C case-sensitive alphabetic characters. Note that $R, C < 1000$.

The end of test cases is specified using a dummy test case where either R or C is zero.

Output Format

For each test case, print the shape of the overlapped grids. If there is no overlapping, print the grid itself. If there is more than one overlapping possibility, print the arrangement with the largest number of shared letters. If there is more than one arrangement with the largest number of shared letters, print any one of them.

After the output of each test case, print a line made of x '+' characters where x is equal to the number of columns used to print the output of that test case.

Sample Input/Output

overlap.in

```
4 6
ABCDEF
GHABCD
IJGHAB
KLIJGH
2 2
ab
AB
3 4
abpq
pqrs
rsab
0 0
```

OUTPUT

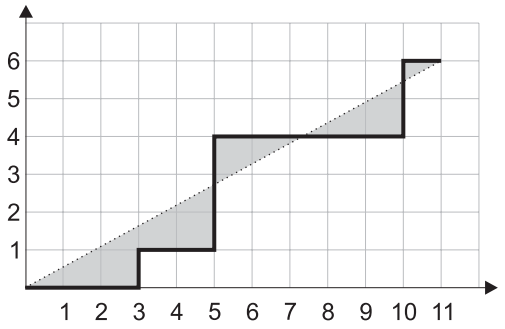
```
ABCDEF
GHABCDEF
IJGHABCD
KLIJGHAB
  KLIJGH
++++++++
ab
AB
++
  abpq
abpqrs
pqrsab
rsab
++++++
```

[E] Geometry?! Why Not??

Program:	`geometry.(c|cpp|java)`
Input:	`geometry.in`
Balloon Color:	Purple

Description

Consider a 2D path drawn in the following manner: Starting at the origin point, we can move only up or right. The path will be described as a string made of zero or more {'U','R'} letters. For each 'U' we'll move one unit up, while 'R' moves one unit to the right. In the following figure, the path constructed by the string RRRURRUUURRRRRUUR is drawn in a thick line.



Imagine now that we draw a straight line that connects the origin point to the last point in the path. (The line that is drawn in dots in the figure above.) We want to compute the total area that falls between the straight line and the path (the grayed area in the above figure.)

Input Format

Your program will be tested on one or more test cases. Each test case is described on a separate line. The path of each test case is described as a string made of one or more {'U','R'} letters and terminated by letter 'S'.

All the test-cases paths can be drawn on a grid of size $1,000 \times 1,000$.

The last line of the input file is made of a single 'S' character and is not part of the test cases.

Output Format

For each test case, write the result on a separate line using the following format:

`k.␣a`

where `k` is the test case number (starting from 1.) `a` is the area rounded to three decimal places.

Sample Input/Output

geometry.in

```
RRRURRUUURRRRRUURS
RUURS
S
```

OUTPUT

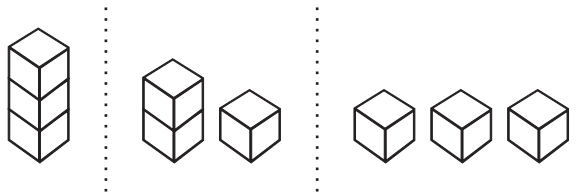
```
1. 8.515
2. 1.000
```

[F] Indistinguishable Blocks

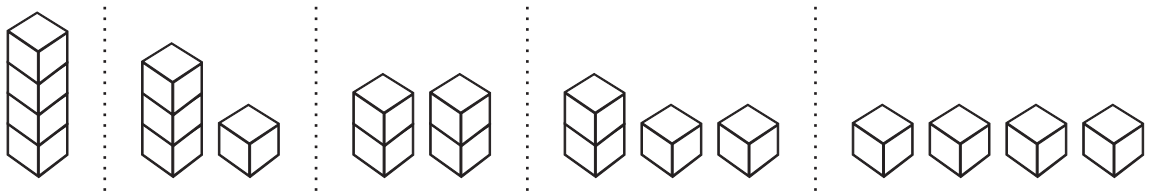
Program:	`blocks.(c|cpp|java)`
Input:	`blocks.in`
Balloon Color:	Gold

Description

In how many ways can you arrange N identical blocks (cubes) into piles. For example, for $N = 3$ there are three arrangements:



Remember, the blocks are identical. For $N = 4$ there are 5 arrangements:



Write a program that determines how many arrangements exist for a given N.

Input Format

Your program will be tested on one or more test cases. Each test case will be specified as a single natural number (denoting `N`, the number of indistinguishable blocks) on a separate line. The end of the test cases is identified by a negative number (which is not part of the test cases.)

Note that $0 \leq N \leq 120$.

Output Format

For each test case, your program should print the result using the following format:

`N␣=>␣P`

Where `N` is the number of indistinguishable blocks for the test case, and `P` is the result.

Sample Input/Output

blocks.in

```
1
3
5
-1
```

OUTPUT

```
1 => 1
3 => 3
5 => 7
```

[G] Chop Ahoy!

Program:	chop.(c|cpp|java)
Input:	chop.in
Balloon Color:	Silver

Description

Given two numbers, say 506 and 119, we want to determine if it is possible to transform the first into the second using the following process: Start with the first number, chopping it up any way you like, for example, 50 and 6 (hereafter written as 50.6.) Now square the pieces and add them up ($50^2 + 6^2 = 2536$.) Now repeat the same process on the result. Say we decide to chop it into 2.53.6 giving us the number 2849. At each step, chop the resulting number any way you like, into one or more parts, then add the squares of the parts. As shown below, in 5 steps, we can transform 506 into 119.

$$50.6 \Rightarrow\ 2.53.6 \Rightarrow\ 2.849 \Rightarrow\ 72.0.80.5 \Rightarrow\ 1.1.6.0.9 \Rightarrow\ 119$$

We could have, however, transformed 506 into 119 in just four steps as shown below:

$$50.6 \Rightarrow\ 2536 \Rightarrow\ 64.31.2.96 \Rightarrow\ 1.4.2.7.7 \Rightarrow\ 119$$

Write a program the takes two natural numbers `S` and `D` and determine the minimum number of steps to transform `S` into `D`. Your program should print -1 if the transformation cannot be performed in 8 or less steps. Intermediate numbers cannot be greater than or equal to 10,000,000.

Input Format

Your program will be tested on one or more test cases. Each test case is specified on a single line containing two natural numbers `S` and `D` separated by a single space character.

Note that $0 \leq S, D < 10,000,000$

The last line of the input file is made of two zeros and is not part of the test cases.

Output Format

For each test case, write the result on a separate line using the following form:

`k.␣result`

where `k` is the test case number (starting from 1.)

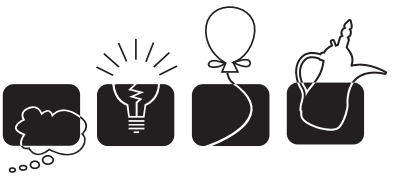
Sample Input/Output

chop.in

```
506 119
2003 2004
0 0
```

OUTPUT

```
1. 4
2. 5
```



[H] Give me a break!

Program:	`cut.(c|cpp|java)`
Input:	`cut.in`
Balloon Color:	Black

Description

Did you ever notice that some words in English can be broken into two or more words? Take for example the word *'Sunday'* which can be broken to *'sun'* and *'day'*. Other examples include: airbag *(air.bag)*, adjust *(ad.just)*, and weathering *(we.at.he.ring)*.

Write a program that reads a dictionary of words, and prints the number of words in that dictionary that can be broken into two or more sub-words (all within the same dictionary) where no sub-word is shorter than 3 letters. Case is insignificant.

Input Format

Your program will be tested on one or more test cases. The dictionary of each test case will be given as a list of words with each word specified on a separate line.

The end of a dictionary is indicated with a line made of '-' characters except the dictionary of the last test case which will end with a line made of '+' characters.

Each dictionary has at least one word but no more than 50,000 words. Each word is at least one character long but no longer than 16 characters. All words will be made of alphabetic characters only.

Output Format

For each dictionary, write the result on a separate line.

Sample Input/Output

cut.in

```
bag
sun
day
moon
Sunday
Monday
airbag
MoonBag
----
straw
black
blue
berry
raspberry
strawberry
blueberry
blackberry
cranberry
HalleBerry
+++
```

OUTPUT

```
2
3
```

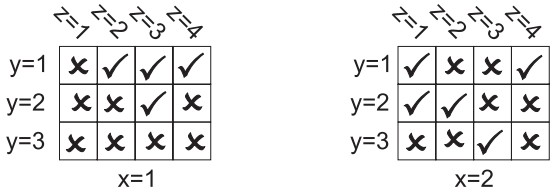
[I] One for all. All for there exists.

Program:	`forall.(c|cpp|java)`
Input:	`forall.in`
Balloon Color:	White

Description

Given a boolean function of N variables, write a program to determine the validity of queries involving universal (for-all) and existential (there-exists) quantifications.

Consider, for example, the following boolean function of three variables x, y, and z:



Here's a sample of queries to answer:

$\forall x \, \forall y \, \forall z$
answer is false, since not every combination is true.

$\exists y \, \forall z \, \exists x$
Answer is true. For `y=1`, we have a value for `x` giving a true result for every value of `z`: `<z=1,x=2>` and `<z=2,x=1>` and `<z=3,x=1>` and `<z=4,x=1|2>`

$\exists x \, \forall z \, \exists y$
Answer is true for `x=2`.

Input Format

Your program will be tested on a number of test cases. Each test case is specified using Q+3 lines.

The first two lines of a test case specify the boolean function while the remaining lines specify the queries for that test case. The first line includes $N + 1$ positive numbers. The first number is N which is the number of variables. The remaining N numbers are $D_1, D_2, \ldots D_N$ where D_i is the number of values for v_i (the i^{th} variable.) The second line is made of $D_1 * D_2 * \ldots * D_N$ `'T'` or `'F'` characters (true or false) specifying the values of the function, in row major order.

The third line includes a single positive value Q representing the number of queries. Each query is specified on a separate line using N pairs, one for each variable (all queries will involve all variables.) Each pair is described using two parts: The first part represents the quantification and is one of two letters: `'A'` (for all) and `'E'` (there exists.) The second part is a number representing the variable number. For example, the query `A2E1` stands for $\forall v_2 \, \exists v_1$

Consecutive test cases are separated by a single blank line. The end of test cases is specified using a dummy test case with N=0.

The number of variables is less than 10. The number of values for any variable will not exceed 10. However, $D_1 * D_2 * \ldots * D_N$ will not exceed $1,000,000$.

The first test case in the Sample I/O section represents the example used in the problem description.

Output Format

For each test case, print the following line:

`Test␣case␣#␣k`

where k is the test case number (starting at 1.) For each query, print the following line:

`␣␣Query␣#␣k␣is␣r.`

where k is the query number (starting at 1) and r is the result of the query (true or false.)

Sample Input/Output

forall.in

```
3 2 3 4
FTTTFFTFFFFFTFFTTTFFFFTF
3
A1A2A3
E2A3E1
E1A3E2

2 2 2
TFTF
6
A2A1
A2E1
A1E2
E1A2
E2A1
E1E2

0
```

OUTPUT

```
Test case # 1
  Query # 1 is false.
  Query # 2 is true.
  Query # 3 is true.

Test case # 2
  Query # 1 is false.
  Query # 2 is false.
  Query # 3 is true.
  Query # 4 is false.
  Query # 5 is true.
  Query # 6 is true.
```