1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

The Second Battle of Waterloo

held at

The University of Waterloo
11-12 November 1994

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Some rules and advice

1. All questions require you to read the test data from standard input and write results to standard output.
   - Output must correspond *exactly* to the provided sample output, including spelling and spacing.
   - All lines should be terminated with a new-line, and no whitespace should appear at the end of a line unless explicitly specified.
   - Tabs should never be used.
2. All programs will be re-compiled prior to testing with the judges' data. Non-standard libraries may not be used in your solutions. You are also not allowed to include any files in your program. The following files will be included standard in all C programs:
   `stdio.h`, `math.h`, `ctype.h`, `strings.h`, and `string.h`.
3. Programming style is not considered in this contest. You are free to code in whatever style you prefer.
4. The CPU time limit for all questions is the same and will be announced before the contest.
5. All questions regarding the contest material should be submitted to the judges through the `submit` command. You can ask the helpers in the room for any non-contest related matters or for getting your printer output. The helpers will not answer any questions regarding the contest material.
6. Judges' decisions are to be considered final. No cheating will be tolerated.

Success!

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Problem A

Lining Up

"How am I ever going to solve this problem?" said the pilot.

Indeed, the pilot was not facing an easy task. She had to drop packages at specific points scattered in a dangerous area. Furthermore, the pilot could only fly over the area once in a straight line, and she had to fly over as many points as possible. All points were given by means of integer coordinates in a two-dimensional space. The pilot wanted to know the largest number of points from the given set that all lie on one line. Can you write a program that calculates this number? Your program has to be efficient!

Input:
The input consists of N pairs of integers, where $1 < N < 700$. Each pair of integers is separated by one blank and ended by a new-line character. The list of pairs is ended with an end-of-file character. No pair will occur twice.

Output:
The output consists of one integer representing the largest number of points that all lie on one line.

Sample Input:

```
1 1
2 2
3 3
9 10
10 11
```

Sample Output:

```
3
```

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Problem B

Simply Syntax

In the land of Hedonia the official language is Hedonian. A Hedonian professor had noticed that many of her students still did not master the syntax of Hedonian well. Tired of correcting the many syntactical mistakes, she decided to challenge the students and asked them to write a program that could check the syntactical correctness of any sentence they wrote. Similar to the nature of Hedonians, the syntax of Hedonian is also pleasantly simple. Here are the rules:

0. The only characters in the language are the characters **p** through **z** and **N**, **C**, **D**, **E**, and **I**.
1. Every character from **p** through **z** is a correct sentence.
2. If s is a correct sentence, then so is **N**s.
3. If s and t are correct sentences, then so are **C**st, **D**st, **E**st, and **I**st.
4. Rules 0. to 3. are the only rules to determine the syntactical correctness of a sentence.

You are asked to write a program that checks if sentences satisfy the syntax rules given in Rule 0.–Rule 4.

Input:
The input consists of a number of sentences consisting only of characters **p** through **z** and **N**, **C**, **D**, **E**, and **I**. Each sentence is ended by a new-line character. The collection of sentences is terminated by the end-of-file character. If necessary, you may assume that each sentence has at most 256 characters and at least 1 character.

Output:
The output consists of the answers **YES** for each well-formed sentence and **NO** for each not-well-formed sentence. The answers are given in the same order as the sentences. Each answer is followed by a new-line character, and the list of answers is followed by an end-of-file character.

Sample Input:

```
Cp
Isz
NIsz
Cqpq
```

Sample Output:

```
NO
YES
YES
NO
```

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Problem C

TeX Quotes

TeX is a typesetting language developed by Donald Knuth. It takes source text together with a few typesetting instructions and produces, one hopes, a beautiful document. Beautiful documents use “ and ” to delimit quotations, rather than the mundane " which is what is provided by most keyboards. Keyboards typically do not have an oriented double-quote, but they do have a left-single-quote ` and a right-single-quote '. Check your keyboard now to locate the left-single-quote key ` (sometimes called the “backquote key”) and the right-single-quote key ' (sometimes called the “apostrophe” or just “quote”). Be careful not to confuse the left-single-quote ` with the “backslash” key \. TeX lets the user type two left-single-quotes `` to create a left-double-quote “ and two right-single-quotes '' to create a right-double-quote ”. Most typists, however, are accustomed to delimiting their quotations with the un-oriented double-quote ".

If the source contained
"To be or not to be," quoth the bard, "that is the question."
then the typeset document produced by TeX would not contain the desired form:
“To be or not to be,” quoth the bard, “that is the question.”
In order to produce the desired form, the source file must contain the sequence:
``To be or not to be,'' quoth the bard, ``that is the question.''

You are to write a program which converts text containing double-quote (") characters into text that is identical except that double-quotes have been replaced by the two-character sequences required by TeX for delimiting quotations with oriented double-quotes. The double-quote (") characters should be replaced appropriately by either `` if the " opens a quotation and by '' if the " closes a quotation. Notice that the question of nested quotations does not arise: The first " must be replaced by ``, the next by '', the next by ``, the next by '', the next by ``, the next by '', and so on.

Input will consist of several lines of text containing an even number of double-quote (") characters. Input is ended with an end-of-file character. The text must be output exactly as it was input except that:

- the first " in each pair is replaced by two ` characters: `` and
- the second " in each pair is replaced by two ' characters: ''.

Sample Input:

```
"To be or not to be," quoth the Bard, "that
is the question".
The programming contestant replied:  "I must disagree.
To `C' or not to `C', that is The Question!"
```

Sample Output:

```
``To be or not to be,'' quoth the Bard, ``that
is the question''.
The programming contestant replied:  ``I must disagree.
To `C' or not to `C', that is The Question!''
```

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Problem D

Jack Straws

In the game of Jack Straws, a number of plastic or wooden "straws" are dumped on the table and players try to remove them one-by-one without disturbing the other straws. Here, we are only concerned with if various pairs of straws are connected by a path of touching straws. You will be given a list of the endpoints for some straws (as if they were dumped on a large piece of graph paper) and then will be asked if various pairs of straws are connected. Note that touching is connecting, but also two straws can be connected indirectly via other connected straws.

Input:
A problem consists of multiple lines of input. The first line will be an integer n $(1 < n < 13)$ giving the number of straws on the table. Each of the next n lines contain 4 *positive integers*, x_1, y_1, x_2 and y_2, giving the coordinates, $(x_1, y_1), (x_2, y_2)$ of the endpoints of a single straw. All coordinates will be less than 100. (Note that the straws will be of varying lengths.) The first straw entered will be known as straw #1, the second as straw #2, and so on. The remaining lines of input (except for the final line) will each contain two positive integers, a and b, both between 1 and n, inclusive. You are to determine if straw a can be connected to straw b. When $a = 0 = b$, the input is terminated. There will be no illegal input and there are no zero-length straws.

Output:
You should generate a line of output for each line containing a pair a and b, except the final line where $a = 0 = b$. The line should say simply "`CONNECTED`", if straw a is connected to straw b, or "`NOT CONNECTED`", if straw a is not connected to straw b. For our purposes, a straw is considered connected to itself.

Sample Input:

```
7
1 6 3 3
4 6 4 9
4 5 6 7
1 4 3 5
3 5 5 5
5 2 6 3
5 4 7 2
1 4
1 6
3 3
6 7
2 3
1 3
0 0
```

Sample Output:

```
CONNECTED
NOT CONNECTED
CONNECTED
CONNECTED
NOT CONNECTED
CONNECTED
```

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

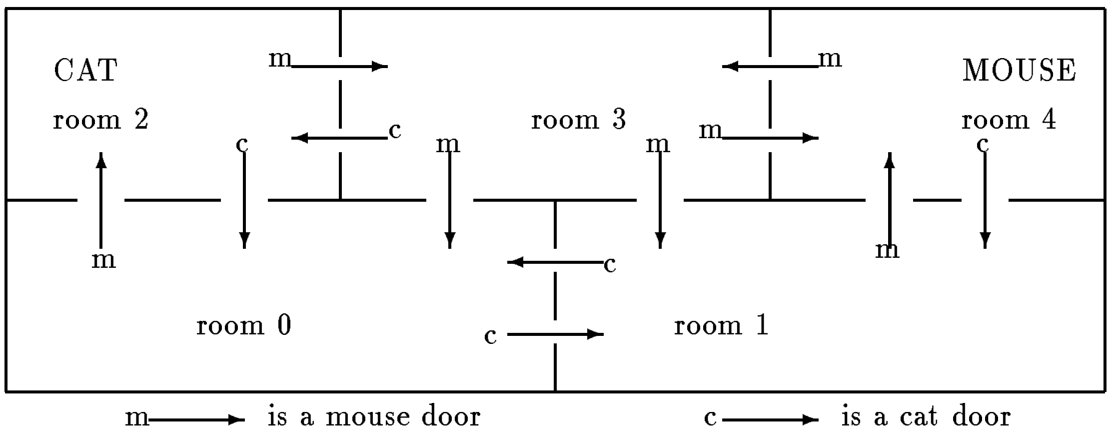
Problem E

Cat and Mouse

In a house with many rooms live a cat and a mouse. The cat and the mouse each have chosen one room as their "home". From their "home" they regularly walk through the house. A cat can go from room A to room B if and only if there is a cat door from room A to room B. Cat doors can only be used in one direction. Similarly a mouse can go from room A to room B if and only if there is a mouse door from room A to room B. Also mouse doors can be used in only one direction. Furthermore, cat doors cannot be used by a mouse, and mouse doors cannot be used by a cat.

Given a map of the house you are asked to write a program that finds out

1. if there exist walks for the cat and mouse where they meet each other in some room, and
2. if the mouse can make a walk through at least two rooms, end in its "home" room again, and along the way cannot ever meet the cat. (Here, the mouse may not ever meet the cat, whatever the cat does.)



For example, in the map, the cat can meet the mouse in rooms 0, 1, and 2. Also, the mouse can make a walk through two rooms without ever meeting the cat, viz., a round trip from room 4 to 3 and back.

Input:
The input consists of integers and defines the configuration of the house. The first line has three integers separated by blanks: the first integer defines the number of rooms, the second the initial room of the cat (the cat's "home"), and the third integer defines the initial room of the mouse (the mouse's "home"). Next there are zero or more lines, each with two positive integers separated by a blank. These lines are followed by a line with two -1's separated by a blank. The pairs of positive integers define the cat doors. The pair A B represents the presence of a cat door from room A to room B. Finally there are zero or more lines, each with two positive integers separated by a blank. These pairs of integers define the mouse doors. Here, the pair A B represents the presence of a mouse door from room A to room B.

The number of rooms is at least one and at most 100. All rooms are numbered consecutively starting at 0. You may assume that all positive integers in the input are legal room numbers.

Output:
The output consists of two characters separated by a blank and ended by a new-line character. The first character is **Y** if there exist walks for the cat and mouse where they meet each other in some room. Otherwise, it is **N**. The second character is **Y** if the mouse can make a walk through at least two rooms, end in its "home" room again, and along the way cannot ever meet the cat. Otherwise, it is **N**.

Sample Input:
(From example above)

```
5 2 4
0 1
1 0
2 0
3 2
4 1
-1 -1
0 2
1 4
2 3
3 0
3 1
3 4
4 3
```

Sample Output:

```
Y Y
```

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Problem F

Expanding Fractions

In this problem you are to print the decimal expansion of a quotient of two integers. As you well know, the decimal expansions of many integer quotients result in decimal expansions with repeating sequences of digits. You must identify these. You will print the decimal expansion of the integer quotient given, stopping just as the expansion terminates or just as the repeating pattern is to repeat itself *for the first time.* If there is a repeating pattern, you will say how many of the digits are in the repeating pattern.

Input:
There will be multiple input instances, each instance consists of two positive integers on a line. The first integer represents the numerator of the fraction and the second represents the denominator. In this problem, the numerator will always be less than the denominator and the denominator will be less than 1000. Input terminates when numerator and denominator are both zero.

Output:
For each input instance, the output should consist of the decimal expansion of the fraction, *starting with the decimal point.* If the expansion terminates, you should print the complete decimal expansion. If the expansion is infinite, you should print the decimal expansion up to, but not including the digit where the repeated pattern first repeats itself. For instance, $4/11 = .3636363636\ldots$, should be printed as `.36`. (Note that the shortest repeating pattern should be found. In the above example, `3636` and `363636`, among others, are repeating patterns, but the shortest repeating pattern is `36`.) Since some of these expansions may be quite long, multiple line expansions should each contain exactly 50 characters on each line (except the last line, which, of course, may be shorter) — that includes the beginning decimal point. (Helpful hint: The number of digits before the pattern is repeated will never be more than the value of the denominator.)

On the line immediately following the last line of the decimal expansion there should be a line saying either "`This expansion terminates.`", or "`The last` n

`digits repeat forever.`", where n is the number of digits in the repeating pattern.

Output for each input instance (including the last input instance) should be followed by a blank line.

Sample Input:

```
3 7
345 800
112 990
53 122
0 0
```

Sample Output:

```
.428571
The last 6 digits repeat forever.

.43125
This expansion terminates.

.113
The last 2 digits repeat forever.

.43442622950819672131147540983606557377049180327486
885245901639
The last 60 digits repeat forever.
```

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Problem G

Egyptian Multiplication

In 1858, A. Henry Rhind, a Scottish antiquary, came into possession of a document which is now called the Rhind Papyrus. Titled "Directions for Attaining Knowledge into All Obscure Secrets", the document provides important clues as to how the ancient Egyptians performed arithmetic.

There is no zero in the number system. There are separate characters denoting ones, tens, hundreds, thousands, ten-thousands, hundred-thousands, millions and ten-millions. For the purposes of this problem, we use near ASCII equivalents for the symbols:

- `|` for one (careful, it's a vertical line, not 1)
- `n` for ten
- `9` for hundred
- `8` for thousand
- `r` for ten-thousand

(The actual Egyptian hieroglyphs were more picturesque but followed the general shape of these modern symbols. For the purpose of this problem, we will not consider numbers greater than 99,999.)

Numbers were written as a group of ones preceded in turn by groups of tens, hundreds, thousands and ten-thousands. Thus our number 4,023 would be rendered: `||| nn 8888`. Notice that a zero digit is indicated by a group consisting of none of the corresponding symbol. The number 40,230 would thus be rendered: `nnn 99 rrrr`. (In the Rhind Papyrus, the groups are drawn more picturesquely, often spread across more than one horizontal line; but for the purposes of this problem, you should write numbers all on a single line.)

To multiply two numbers a and b, the Egyptians would work with two columns of numbers. They would begin by writing the number `|` in the left column beside the number a in the right column. They would proceed to form new rows by doubling the numbers in both columns. Notice that doubling can be effected by copying symbols and normalizing by a carrying process if any group of symbols is larger than 9 in size. Doubling would continue as long as the number in the left column does not exceed the other multiplicand b. The numbers in the first column that summed to the multiplicand b were marked with an asterisk. The numbers in the right column alongside the asterisks were then added to produce the result. Below, we show the steps corresponding to the multiplication of 483 by 27:

```
| *                                     ||| nnnnnnnn 9999
|| *                                    |||||| nnnnnn 999999999
||||                                    || nnn 999999999 8
|||||||| *                              |||| nnnnnn 99999999 888
|||||| n *                              |||||||| nn 9999999 8888888
The solution is:  | nnnn 888 r
```

(The solution came from adding together:

```
||| nnnnnnnn 9999
|||||| nnnnnn 999999999
|||| nnnnnn 99999999 888
|||||||| nn 9999999 8888888.)
```

You are to write a program to perform this Egyptian multiplication.

Input:

Input will consist of several pairs of nonzero numbers written in the Egyptian system described above. There will be one number per line; each number will consist of groups of symbols, and each group is terminated by a single space (including the last group). Input will be terminated by a blank line.

Output:

For each pair of numbers, your program should print the steps described above used in Egyptian multiplication. Numbers in the left column should be flush with the left margin. Each number in the left and right column will be represented by groups of symbols, and each group is terminated by a single space (including the last group). If there is an asterisk in the left column, it should be separated from the end of the left number by a single space. Up to the 40th character position should then be filled with spaces. Numbers in the right column should begin at the 41st character position on the line and end with a newline character. Test data will be chosen to ensure that no overlap can occur. After showing each of the doubling steps, your program should print the string: "`The solution is:  `" followed by the product of the two numbers in Egyptian notation.

Sample Input:

```
||
||
|||
||||
nnnnnn 9
||| n
n
9
|||
8
```

Sample Output:

```
|
|| *
The solution is: ||||
|
||
|||| *
The solution is: || n
| *
||
|||| *
|||||||| *
The solution is: nnnnnnnn 88
|
||
|||| *
||||||||
|||||| n
|| nnn *
|||| nnnnnn *
The solution is: 8
|
||
||||
|||||||| *
|||||| n
|| nnn *
|||| nnnnnn *
|||||||| nn 9 *
|||||| nnnnn 99 *
|| n 99999 *
The solution is: 888
||
||||

|||
||||||
|| n

nnnnnn 9
nn 999
nnnn 999999
nnnnnnnn 99 8

n
nn
nnnn
nnnnnnnn
nnnnnn 9
nn 999
nnnn 999999

|||
||||||
|| n
|||| nn
|||||||| nnnn
|||||| nnnnnnnnn
|| nnnnnnnnn 9
|||| nnnnnnnn 999
|||||||| nnnnnn 9999999
|||||| nnn 99999 8
```

1994 East-Central Regionals
of the ACM International Collegiate Programming Contest

Problem H

Cabinets

Well-Built Cabinet Distributors, Inc. recently received an electronic catalog of cabinets from their leading manufacturer, Woodcraft. Unfortunately, the format of the data is not consistent with that expected by Well-Built's inventory software. For this problem, you will construct a program that reformats the Woodcraft catalog for use by the inventory software.

Input:
Input will be formatted as from a comma-delimited ASCII file. Your program must read all input from the standard input file. Each line in the input has a maximum length of 64 characters and contains the following fields:

Field	Length	Explanation
Style Code	1-3	Code specifying the cabinet's style.
Style Name	0-15	Name for cabinet style.
Description	1-15	Code describing type of cabinet.
Extension	0-25	Additional information about cabinet.
Unit Price	0-6	Manufacturer's suggested retail price (dollars x 100).

The records are presented in ascending order by Style Code. You may assume that all fields will be consistent with the lengths given, and that all fields will contain appropriate characters.

Output:
Output will consist of a reformatted catalog. The reformatted catalog file will be a comma-delimited ASCII file. The first record in the file must be the following:

```
Item Id,Item Desc,Item Price
```

Each remaining record in the file will have a maximum length of 50 characters and contain the following fields:

Field	Length	Explanation
Item Id	4-13	Unique identification code for inventory database.
Item Desc	1-30	Inventory description of cabinet.
Item Price	4-7	Manufacturer's suggested retail price.

The Item Id is formed by concatenating the Style Code and the Description. If the Style Code is less than three characters long, it must be left-filled with zeros to three characters. If the Item Id exceeds 13 characters, then the record is rejected.

The Item Desc is formed by concatenating the Style Name, a hyphen, and the Extension. If the Style Name is missing, use the Style Name from the first record of the corresponding Style Code group. If this first record has no Style Name either, then reject the record. If the Extension is not present, then Item Desc is the same as Style Name (no hyphen). If Item Desc exceeds 30 characters, then it must be truncated on the right.

The Item Price is formed by formatting the Unit Price as dollars and cents. If Unit Price is not present, then Item Price = 0.00.

Sample Input:

```
23,CHAMPAGNE,BASE36,3" RECESSED TOE KICK,8900
23,,BASE54,,11000
25,LAUREL,CNR24LT,,15000
107,COLONIAL,BASE54WSIDEJAM
202,SAGEBRUSH,OVRHD54P,USE WITH HDWARE KIT #3207
221,ALVEA MODERN,BASE36
221,ALVEA MODERN,OVRHD54WCAP
```

Sample Output:

```
Item Id,Item Desc,Item Price
023BASE36,CHAMPAGNE-3" RECESSED TOE KICK,89.00
023BASE54,CHAMPAGNE,110.00
025CNR24LT,LAUREL,150.00
202OVRHD54P,SAGEBRUSH-USE WITH HDWARE KIT ,0.00
221BASE36,ALVEA MODERN,0.00
```