Problem 1 - Fort Knox has been robbed!

The perpetrators of this heinous act slipped away with billions of dollars in gold bullion, in this, the largest theft in the nation's history.

It is unknown exactly who the criminals were, but based on a comprehensive investigation as well as tips from various confidential sources, the FBI has compiled a list of suspects.

It is believed with certainty that to have planned, tested, and successfully executed such a slick and efficient operation, the perpetrators had to have lived in the same city for a long period of time. Additionally, it is believed that there were no less than 3 individuals involved, and chances are, there were many more.

Following is an excerpt of the FBI's current suspect list. This unordered list includes where each suspect lived and from what dates. You can use this data to develop your program which, once finalized, will be used on the entire suspect list to narrow down the search, and ultimately bring these criminals to justice!

Example Input:

```
OSCAR,Morocco,10/02/2006,05/18/2007
OSCAR,Casablanca,05/19/2007,08/26/2009
XENA,Morocco,01/01/2005,10/10/2006
TILULA,Hattiesburg,03/15/1990,09/15/2001
NATHAN,Morocco,05/10/2002,09/15/2010
XENA,Casablanca,01/22/1991,04/04/2003
TILULA,Morocco,08/19/2006,11/10/2010
NATHAN,Hattiesburg,12/01/1988,12/01/2001
```

Your mission (if you choose to accept it) is to develop a program that will determine which suspects lived in the same city, at the same time.

The program's output should consist of the following data columns in the following order (do not include column names/headers):

1. Suspects – a concatenated/delimited list of suspects that lived in the same city for a period of time, sorted by name (ascending). This suspect name list should be delimited by a semi-colon [;].
2. City name – the name of the city the suspects commonly lived in.
3. Start date – the date the group of suspects began living in the common city.
4. End date – the date the group of suspects stopped living in the common city.

Each output record should be:

1. Unique.
2. Internally delimited by a comma [,] (each data column delimited by a comma).
3. Terminated by a newline character.
4. Primarily sorted by the number of perpetrators (descending).
5. Within the primary sort, sorting should be done on the suspects, city, start date, and end date columns (ascending). For example, if there are multiple records with 4 perpetrators, the records in this set should be sorted across the row (suspects, city, start date, end date).

To recap:

- Within each row order the suspect names in ascending order.
- Group by the number of suspects so that all rows with four suspects are together, those with three suspects are together, those with two suspects are together, etc.
- Within the group sort by the suspects column first, then by the city column, the start date column, and lastly by the end date column.

Other information:

1. Suspects can only live in one place at one time.
2. Account for leap years if necessary (standard Gregorian style).
3. Suspects' names and cities are unique (e.g. 2 different suspects will not be named "Bob", and 2 different cities going by the name "Paris" will not be used).
4. When sorting characters, all characters should be treated as uppercase.
5. Dates are/should be in standard MM/DD/YYYY format.
6. All text in the output should be in the same case it was in in the input.
7. Gaps may exist between time periods of residence for an individual (e.g. all residence time periods for an individual may not be contiguous)
8. Multiple, fully contiguous time periods in the same city will not exist, though people are known to move back to a location after living in another for a while...

Example Output:

```
NATHAN;OSCAR;TILULA;XENA,Morocco,10/02/2006,10/10/2006
NATHAN;OSCAR;TILULA,Morocco,10/02/2006,05/18/2007
NATHAN;TILULA;XENA,Morocco,08/19/2006,7/10/2006
NATHAN;TILULA;XENA,Morocco,08/19/2006,10/10/2006
OSCAR;TILULA;XENA,Cairo,11/02/2006,10/10/2006
OSCAR;TILULA;XENA,Morocco,09/02/2006,10/10/2006
OSCAR;TILULA;XENA,Morocco,10/02/2006,10/10/2006
```

Problem 2 - Telegraph Printer

You must implement a telegraphic printer. It consists of a "daisy wheel" made of the characters A-Z (all uppercase), 0-9, period, and space, in that order. There are two inputs. One input causes the daisy wheel to turn one notch (for example, from A to B). The other input tells the printer to print the current character.

Input will be provided as lines with three columns each. The first column represents the signal to turn the wheel. The second column is just a separator and will always contain a single space. The third column represents the signal to print. If the first column contains a "." character then the wheel must turn one notch. If the third column contains a "." then the current character is printed. Each line of input will only contain a single "." character. A line with a single "@" character signifies the end of the input.

The input assumes the wheel begins with "A" being the selected character. The wheel maintains its position when a character is printed.

Sample Input

```
.
.
.
.
.
.
.
  .
.
  .
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
```

```
   .
 .
 .
 .
 .
 .
 .
 .
 .
 .
 .
 .
   .
@
```

Sample Output

```
HI.
```

Problem 3 - Integer Palindrome

A Palindrome is literary term when a phrase is spelled the same forwards and backwards. "Never odd or even", "Rise to Vote Sir", and "Go hang a salami; I'm a lasagna hog" are examples of palindromes. A palindrome may be an integer as well. For example: 5, 11, 343, 54345 are considered palindromes because their digits can be read backwards and forwards.

A numeral system (or system of numeration) is a writing system for expressing numbers, that is a mathematical notation for representing numbers of a given set, using graphemes or symbols in a consistent manner. For example, base 10 are represented by digits 0-9, base 2 (or binary) contain digits 0 and 1.

The Task is to take a given positive base-10 number, convert that number to another base system, and see if that number is a palindrome in the other base system.

Key Points

- A check number is the base-10 number that is to be transformed to the base specified by the second input parameter.
- The check number will always be in base 10.
- A base number is an integer that represents how many numbers used in a number system. For example, base 2 would be represented as 1s and 0s (binary); and Base ten is a 0-9 number system.
- For any base systems greater than 10 (i.e. Base 27), extra digit values are represented (in order) by letters A-Z, followed by a-z. Base 16, for example will be represented by (0-9 then A-F, with a base 16 value of F representing the base 10 equivalent value of 15).
- The check number will be converted to a base system that is represented by the base number.
- Once the check number is converted the palindrome test will be conducted.
- Error checking will be done on each pair of numbers prior to palindrome testing.

Input

The Input will consist of 2 base-10 numbers delimitated by a single space. The first number is the check number, and the second number is the base number.

Output

The output will consist of a line of text that will either show the results of the palindrome test, or report an error. This line of text will have only single spaces between words and after punctuation. If the input line passes the error check, an output of the palindrome test is to be made. The allowed outputs must conform to the following (Note '##' below will be replaced by actual values in the final output):

1. ## in base# is ## - Yes
2. ## in base# is ## - No
3. Error: Check number is not a positive base10 number.
4. Error: Base number is not a positive base10 number.
5. Error: Invalid number of arguments. Expected 2 numbers for input.
6. Error: Base ## is an unsupported base system. Maximum Base system is Base 62.

Sample Input

```
100 10
2409 2
A 1
251 37
130 42
1 A
3887 16
12
13 4 12
33 64
```

Sample Output

```
100 in base10 is 100 - No
2409 in base2 is 100101101001 - Yes
Error: Check number is not a positive base10 number
251 in base37 is 6T - No
130 in base42 is 34 - No
Error: Base number is not a positive base10 number.
3887 in base 16 is F2F - Yes
Error: Invalid number of arguments. Expected 2 numbers for input.
Error: Invalid number of arguments. Expected 2 numbers for input.
Error: Base 64 is an unsupported base system. Maximum Base system is Base 62.
```

Problem 4 - Vigenère Cipher

The Vigenère cipher is a method of encrypting text by using a series of Caesar ciphers. A Caesar cipher is a simple substitution cipher which works by rotating the plain text by a fixed number (called a shift). For example, a Caesar cipher with a shift of 3 will replace A with D, B with E, etc.

```
Plain:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:   DEFGHIJKLMNOPQRSTUVWXYZABC
```

The Vigenère cipher takes this a step further and creates a matrix of the 26 possible Caesar ciphers:

	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

This cipher requires the use of a keyword. The letters of the keyword are repeated for the length of the plaintext and the cryptographer looks up the encrypted letter in the Vigenère table to create the cipher text; the keyword letter is the row indicator and the plain text letter is the column indicator. For example, the plaintext "attackatdawn" with the keyword "LEMON" produces the cipher text "LXFOPVEFRNHR":

Plaintext: `attackatdawn`

Keyword: `LEMONLEMONLE`

Cipher text: `LXFOPVEFRNHR`

Your task is to create a Vigenère cipher program which accepts the plain text and its keyword and outputs the cipher text. According to cryptographic tradition, your cipher text should have no punctuation and be in all uppercase.

Key Points

- Remove all punctuation (periods, commas, semicolons, question marks, etc.) and spaces. There will be no numbers in the plain text.

- The keyword will always be shorter than the plain text.
- The final output should be all uppercase.

Input

Input will be provided through stdin. The input will contain multiple sets to encrypt (the plain text and its keyword being a set). The plain text will be on a separate line from the keyword and each set is separated by a newline. A "@" symbol will indicate the end of the file.

Output

The output will consist of the cipher text generated by your program. Each plain text/keyword pair will produce one line of cipher text. There will be a newline between each cipher text result.

Sample Input

```
Hello! How are you?
CAT

Encrypt this text?
FIRE
@
```

Output for the Sample Input

```
JEENOAQWTTERQU
JVTVDXKXMQJXJFK
```

Problem 5 - Coin Combinations

John is counting the loose change found in his pocket, and wonders to himself about the various combinations of coin values that can make up a specific monetary value. You have **infinite** pennies, nickels, dimes, and quarters. To help John out, write a program that will output **all** combinations of coins such that the total is equal to the input cent value.

For example: 3 quarters, 2 nickels, and 4 pennies is one of many ways to make up 99 cents.

Key Points

- The output should begin with a combination of coin values that make up the input cent value, and each new coin combination should begin on a new line.
- Each coin value must be separated by a comma with **no** spaces.
- Each combination output line must end with a coin value (no ending comma).
- The output file should also include the total number of combinations found; for example, "There are # combinations." where # is the number of combinations. See example output for more information.
- The input monetary value must be in **cents**.

Input

The input will consist of one input line delimited by a new line character. The value must be in a cent value. For example, to input $1.50 you would enter 150.

Output

The output will consist of a combination line that is delimited by a new line character. Each coin should be separated by a comma with no spaces. Each combination should be ordered from the lowest coin denomination first to the highest coin denomination last. And the lines should be ordered from longest combination first to shortest combination last. The total number of combinations should be displayed after the combinations are outputted, and spaced with a new line.

Sample Input

```
10
```

Example output

```
1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,5
5,5
10

There are 4 combinations.
```

Problem 6 - Connect 4

Jim Bob is a great Connect 4 player, great defensive Connect 4 player to be more specific. Jim Bob has never lost a game. Unfortunately for ole JB he hasn't ever won a game either. For some reason Jim Bob can see his opponent's winning moves way before they occur and he can always block them. What he can't do is see his own winning moves. Write a program that will analyze the current state of the board and tell Jim Bob if he has a winning move or not. Jim is O and it's Jim's turn to go.

Input

Input will be of multiple boards through stdin. Each board will be separated by an empty line. The @ symbol will indicate the end of input.

Output

For each board read in there will be one line of output. That line will consist of a string value of "Yes" or "No" followed by a space then the column number that Jim Bob should drop his piece in to win. Column numbers are 1-7 from left to right. If there is more than one winning move each winning column number will be printed each separated by a comma.

Sample Input

```
|       |
|       |
|       |
|XO     |
|XXOX   |
|OXXOOO |
---------

|       |
|       |
|       |
| X OO  |
| XOXX  |
|XOXOOOX|
---------

|       |
|       |
|       |
|       |
|       |
| XOXO  |
---------
@
```

Sample Output

```
Yes 1,7
Yes 5
No
```

Problem 7 – Algorithm Fun (The *3n + 1* Problem)

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive, etc.). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

Consider the following algorithm:

```
1. input n
2. print n
3. if n = 1 then STOP
4. if n is odd then n ← 3n + 1
5. else n ← n/2
6. GOTO 2
```

Given the input 22, the following sequence of numbers will be printed:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers *n* such that $0 < n < 1{,}000{,}000$ (and, in fact, for many more numbers than this.)

Given an input *n*, it is possible to determine how many numbers printed before and including the 1 is printed. For a given *n* this is called the *cycle-length* of *n*. In the example above, the cycle length of 22 is 16.

For any two numbers *i* and *j* you are to determine the maximum cycle length over all numbers between and including both *i* and *j*.

The Input

The input will consist of a series of pairs of integers *i* and *j*, one pair of integers per line, separated by a single space. All integers will be less than 10,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including *i* and *j*.

The Output

For each pair of input integers *i* and *j* you must output *i*, *j*, and the maximum cycle length for integers between and including *i* and *j*. These three numbers must be separated by only one space with all three numbers on one line and with one line of output for each line of input. The integers *i* and *j* must appear in the output in the same order in which they appeared in the input and must be followed by the maximum cycle length (on the same line).

Error Checking

Number of parameters per line. Error Message: "Invalid # of parameters on line: *x*"

Input is integers only, must be between 1 and 10,000, and first integer must be less than second integer. Error Message: "Invalid range on line: *x*"

NOTE: There will be a maximum of 1 error for any given line of input. In other words, there will be either a range issue or a parameter count issue. Never both.

Sample Input

```
1 10
100 200
55
201 210
850 12000
33 32
900 1000
1500 1700
-25 700
```

Sample Output

```
1 10 20
100 200 125
Invalid # of parameters on line: 3
201 210 89
Invalid range on line: 5
Invalid range on line: 6
900 1000 174
1500 1700 180
Invalid range on line: 9
```

Problem 8 – What's My Type?

Big Global Distribution ships products coming from and going to a large number of different import/export client companies. Soon they will be performing a data import to a new system. Fortunately, the negotiations between Big Global Distribution and their clients resulted in everyone agreeing on using simple flat files for the data. Unfortunately, no one thought to negotiate the formatting of the files. And now the data is already coming in.

Aloysius, your manager, has tasked your team with composing an application that will analyze the inventory data submitted by the top clients and produce a description of what data types each record contains. Also provide a description of the data types for the entire dataset.

Key Points

- Line feeds and carriage returns are line and record delimiters. They will not be a part of data within the records.
- The pattern of three *at signs* "@@@" (without quotes) on a single line delimits each data set. They will also be present at the end of the last data set.
- The first line of information is the header. The header contains two characters: The *field delimiter* character followed by the *text qualifier* character. Each line following the header is considered a record of data within the entire data set.
- Each record will contain the same number of fields delimited by the *field delimiter* character. The *field delimiter* is the character used in records to separate fields within the record. The *text qualifier* is the character used to surround a field to indicate explicitly that the field is of TEXT datatype.
- The *field delimiter* and the *text qualifier* can be any "printable" ASCII character with decimal values 33 through 126 as well as space (32) and horizontal tab (9) characters.
- Only two data types are defined: NUMERIC and TEXT. A field is considered NUMERIC if it is composed of only numeric characters 0 through 9. Decimals and a leading minus sign are considered numeric characters. A field is considered TEXT if it contains characters that are not numeric and/or if the field is surrounded using *text qualifier* characters.
- Note that not all TEXT fields require *text qualifiers*. The only time a TEXT field requires *text qualifiers* is when the data of the field itself contains a *field delimiter* character or a *text qualifier character*. If a field begins with a *text qualifier* it will always end with a *text qualifier*.

- Fields with embedded characters that are the same as the *text qualifier* character must be surrounded by *text qualifier* characters and the embedded character is represented as a pair of the character.
  The following is an example of a TEXT field with * as the *text qualifier* and embedded characters:
  1000,*TestData*,123,*This field contains **Embedded** characters*,9999

- No input data will be malformed.

Input

Input will be provided through stdin. The input will consist of multiple data sets and will be formatted as described above in the 'Key Points' section. An empty line will signal the end of input.

Output

The output will consist of a row representing each record from the data set indicating what the data type of each field discovered for that record. Each row will be in the following format:

```
Rec #: TEXT,NUMERIC,TEXT
```

- Single space between "Rec" and the record number (starting with 1).
- Single space between the colon and the first field data type.
- New line after the last field data type.

At the end of a data set, output the assumed data type for the field for the entire data set. If a field is the same data type across all records, then the field type for the entire dataset is that type. If a field is of type TEXT for any record, then the field type for the entire dataset is TEXT. Output this line in the following format:

```
Dataset #: TEXT,NUMERIC,TEXT
```

- Single space between "Dataset" and the record number (starting with 1).
- Single space between the colon and the first field data type.
- New line after the last field data type.

Note specifically the last data field of each record and the corresponding data set result in the examples below. Green indicates a TEXT field and blue indicates a NUMERIC field.

Sample Input

```
,"
1000,123,SUPER FUN BALL,SUPER CO.,344 SOMETHING STREET,"SPRINGFIELD, MI 12345",25.23,1112,A32542
1001,3,"""ELEFUNKY"" PLUSH ANIMAL","DUMBO, LLC",1521 STREET RD,"SPRINGFIELD, MI 12345",22.5,1112,"32542"
@@@
|'
1000|123|SUPER FUN BALL|SUPER CO.|344 SOMETHING STREET|'SPRINGFIELD, MI 12345'|25.23|1112|32542
1001|3|'''ELEFUNKY'' PLUSH ANIMAL'|'DUMBO, LLC'|1521 STREET RD|"SPRINGFIELD, MI 12345"|22.5|1112|32542
@@@
,*
1000,123,SUPER FUN BALL,SUPER CO.,344 SOMETHING STREET,*SPRINGFIELD, MI 12345*,25.23,1112,A32542
1001,3,*"ELEFUNKY" PLUSH ANIMAL*,*Q**Bert, LLC*,1521 STREET RD,*SPRINGFIELD, MI 12345*,22.5,1112,32542
@@@
```

Output for the Sample Input

```
Rec 1: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,TEXT
Rec 2: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,TEXT
Dataset 1: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,TEXT
Rec 1: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,NUMERIC
Rec 2: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,NUMERIC
DataSet 2: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,NUMERIC
Rec 1: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,TEXT
Rec 2: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,NUMERIC
DataSet 3: NUMERIC,NUMERIC,TEXT,TEXT,TEXT,TEXT,TEXT,NUMERIC,NUMERIC,TEXT
```

Problem 9 – What's the word?

Joe is a curious individual who enjoys complex problems. He wants to create a program that will allow him to input integers and output those integers written as words.

For instance, if input is 250 then output should be "twohundredfifty".

Input

Input shall consist of one number per line each line is terminated with a new line character

Input will be terminated when no more lines are present

Input must be integers from 1 to 999999999 (one less than a billion)

Output

Output will be the word of the integer entered.

Also, output will be one line of output for each line of input read

Sample Input

```
49
101
102190
234809
1203495
999999999
```

Sample Output

```
fortynine
onehundredone
onehundredtwothousandonehundredninety
twohundredthirtyfourthousandeighthundrednine
onemilliontwohundredthreethousandfourhundredninetyfive
ninehundredninetyninemillionninehundredninetyninethousandninehundredninetynine
```

Problem 10 – Domino Effect

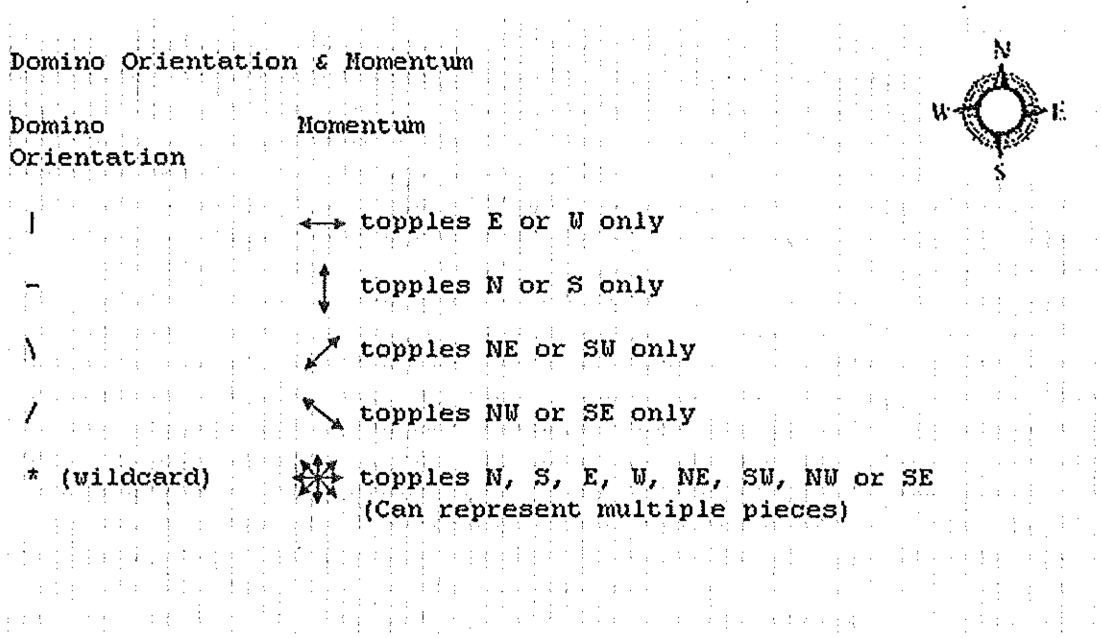
Jeff loves setting up elaborate patterns of dominoes and watching them fall. He spends hours a day designing and testing different patterns. He wants to spend less time doing trial and error and more time creating patterns that work. One day, he has an idea that software could actually save him precious time by allowing him to proof his patterns first.

Jeff needs your help building a program to analyze his pattern of dominoes and provide him feedback to identify potential flaws in his design.

Note: Dominoes are frequently referred to as 'tiles' in this problem.

Key Points

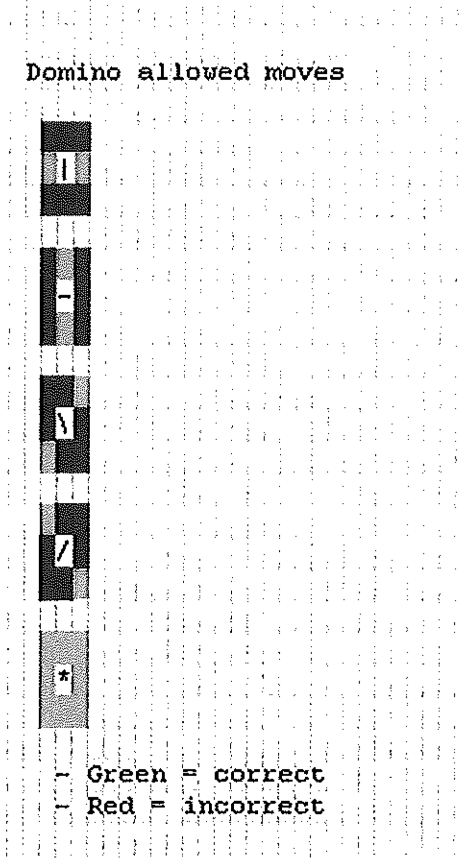
- The letter 'S' marks the start of a pattern. There is only **one** starting point in a pattern.
- Patterns are NOT required to have a single ending point.
- No spaces are allowed between the starting point 'S' and the first tile.
- Tiles are represented in the grid with the following characters: pipe (|), dash (-), backslash (\), forward slash (/) and asterisk (*). ***Please refer to Figure 1 for more information.***
- No spaces are allowed between tiles. Furthermore, a space indicates the end of a run.
- Orientation and momentum should be considered for each falling tile. (i.e. A tile such as '|' is oriented to fall East-West, depending on the previous falling tile's momentum. If the tile is struck on its Eastern side, it should fall to the West.)
- A wildcard tile (indicated by *) is considered to represent several tiles, depending on the pattern layout. In other words, it is used when forming junctions and can trigger multiple runs based on the momentum of the tile striking it. ***Please refer to the example diagrams for more information.***
- A **falling** wildcard tile (indicated by *) can only affect neighboring piece(s) located at angles of 45 degrees or less (relative to the direction it is falling). This means a falling tile cannot fall directly to its right or left (a 90 degree angle).
- Any fallen tile must be changed to the letter 'O'. ***Please refer to the 'Output' section for more information.***



Domino Orientation & Momentum

Domino Orientation	Momentum
|	↔ topples E or W only
-	↕ topples N or S only
\	⤢ topples NE or SW only
/	⤡ topples NW or SE only
* (wildcard)	✳ topples N, S, E, W, NE, SW, NW or SE (Can represent multiple pieces)

Figure 1. Identification of tile characters, their orientation and momentum



Domino allowed moves

- Green = correct
- Red = incorrect

Figure 2. Valid and Invalid moves

Examples

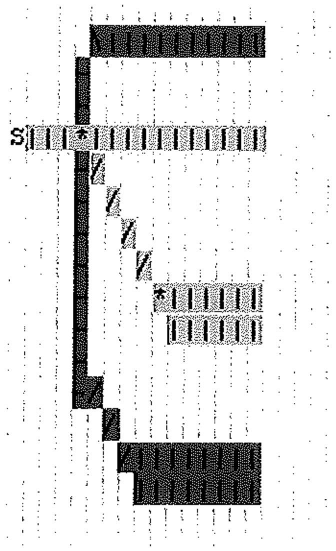


Figure 3. Green represents correct path followed

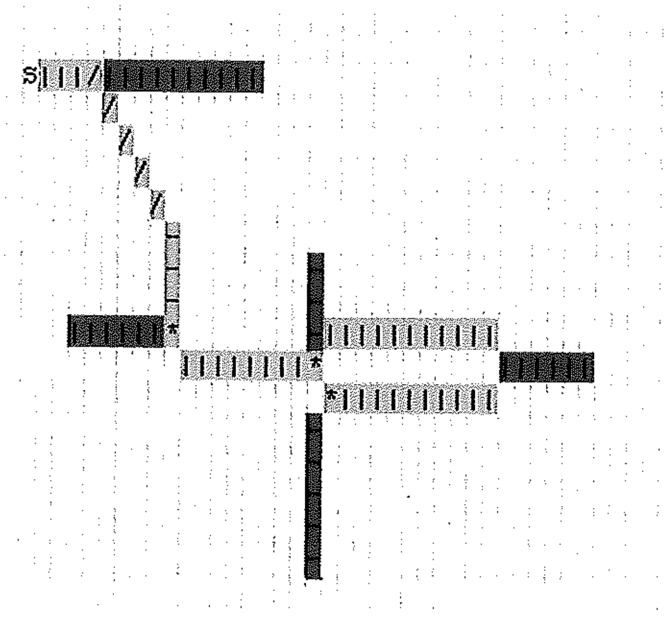


Figure 4. Green represents correct path followed

Input

The input will consist of multiple input lines delimited by a new line character. Each input line can consist of any of the tile characters and spaces as described in the 'Key Points' section. EOF will signal the end of input.

Output

The output will consist of the same exact layout as the input pattern (i.e. It will retain the original spacing and standing tiles.). The **only** difference is the fallen tiles will now be replaced with the letter 'O'. EOF will signal the end of input.

Sample Input

```
s|||/||||||||||
     /
      /
       /
        /
         -
         -        -
         -        -
   ||||||*        -|||||||||||
          ||||||||*           ||||||
                   *||||||||||
                  -
                  -
                  -
                  -
                  -
```

Output for the Sample Input

```
soooo||||||||||
     o
      o
       o
        o
         o
         o        -
         o        -
   ||||||o        -ooooooooooo
          ooooooooo           ||||||
                   ooooooooooo
                  -
                  -
                  -
                  -
                  -
```

Problem 11 - Twin Primes

In mathematics, numbers greater than 1 that have no divisors other than itself and 1 are considered prime. The prime numbers between 1 and 12 are 2, 3, 5, 7 and 11. Prime numbers that are separated by a single even number on a standard number line are considered twin prime pairs. (3,5) and (5,7) are both twin prime pairs.

Write a program that will find the summation of all the twin prime pairs and the number of twin prime pairs between two given integers, including the endpoints. The constraint of the input is that integers given will be between 1 and 10,000.

Input

Input will be two integers per line separated by a single space. There will be a new line character after each line of input. EOF will indicate all input is done.

Output

Output will be the summation of all twin prime pairs between the two given integers on one line. This will be followed by the total number of prime pairs in the given range on the second line.

Sample Input

```
10 100
100 25
```

Sample Output

```
468
6
408
4
```