New Zealand Programming Contest 2021

Problem Set

PREAMBLE

Please note the following very important details relating to input and output:

- Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin`, `Console.ReadLine` or equivalent. Input will be redirected from a file to form the input to your submission.

- Do NOT prompt for input as this will appear in your output and cause a submission to be judged as wrong.

- Write all output to the screen, i.e. use `stdout`, `System.out`, `cout`, `Console.WriteLine` or equivalent. Do not write to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen, such as `conio`, `Crt` or anything similar.

- Output from your program is redirected to a file for later checking. Use of direct I/O means that such output is not redirected and hence cannot be checked. This could mean that a correct program is rejected! You have been warned.

- Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by spaces.

- An *uppercase letter* is a character in the sequence 'A' to 'Z'. A *lower case letter* is a character in the sequence 'a' to 'z'. A *letter* is either a lower case letter or an upper case letter.

- Unless otherwise stated, a *word* or a *name* is a continuous sequence of letters.

- Unless otherwise stated, a *string* is a continuous sequence of visible characters.

- Unless otherwise stated, words and names will contain no more than 60 characters, and strings will contain no more than 250 characters.

- If it is stated that ‘a line contains no more than *n* characters’, this does not include the character(s) specifying the end of line.

- Input files are often terminated by a ‘sentinel’ line, followed by an end of file marker. This line should not be processed.

New Zealand Programming Contest 2021

PROBLEM A **SECURITY CHECK** **3 POINTS**

Keepsafe is a tool used by KiwiBank to "provide an extra layer of security to help protect customers while using Kiwibank Internet Banking." Customers have to set up a number of security questions and answers. After logging in with a user name and password, a customer is presented with one of their security questions and a space for the answer. Two letters in the answer have to be entered as shown by white boxes.

For example:

If the pet's name was "Whisper", the customer would press the P key then the R key. The letters pressed must correspond, in order, to the two spaces in the outline answer to the security question.

For this problem, you have to check customers' answers and verify that they are as expected.

Input

Input consists of a security question answer and a login attempt for one customer, each on a separate line.

The first line contains the security question answer. It will consist of upper case letters and single spaces only, starting and ending with a letter. There will be no more than 32 characters.

The next line shows the login attempt which consists of 2 positive integers followed by 2 upper case letters, all separated by single spaces. The integers are the numbers of the characters required to fill the two spaces in the outlined answer. They are presented in ascending order. When numbering characters in the answer, the first character is 1, but only letters are assigned numbers, spaces being ignored. The two letters are those entered by the customer into the two spaces. They are expected to correspond to the letters in the appropriate places in the security answer required.

Output

Output consists of a single line containing one word. If the two letters entered match the letters in the required places in the answer, in the order presented, then the output will be "`correct`". If one or both letters are not correct, the output is "`error`".

Turn over for sample input and output

Sample Input 1

```
THE ALL BLACKS
4 9 A A
```

Output for Sample Input 1	**Explanation**
`correct`	Ignoring spaces, letters 4 and 9 in THE**A**LLBL**A**CKS are both As.

Sample Input 2

```
WHISPER
1 6 W P
```

Output for Sample Input 2	**Explanation**
`error`	Letters 1 and 6 in **W**HISP**E**R should be W and E, not W and P.

New Zealand Programming Contest 2021

PROBLEM B **HAVE YOU HAD YOUR BIRTHDAY YET?** **3 POINTS**

Today it is 11th September. If you were born before 11th September (in whatever year you were born) then you have already had your 2021 birthday. If you were born after 11th September, you have not yet had your 2021 birthday. If you were born on 11th September – happy birthday! If you were born on 29th February – unlucky, you do not have a birthday this year, as 2021 is not a leap year!

Input

Input will consist of a number of lines containing dates. All dates will be valid and no date will be repeated, so there will be no more than 366 lines. The format for a line will be one or two digits, a space, and the full name of a month. The name will begin with an upper case letter, and the other letters will be lower case. Input will be terminated by a line containing just 0 #. This line should not be processed.

Output

Output will consist of a single line of text for each line of input:

`"You have had your birthday."` if a person born on that date would have already had their birthday in 2021,

`"Your birthday is still to come."` if they would not,

`"Happy birthday!"` if the date is 11th September and

`"Sorry, leapling, no birthday this year."` if the date is 29th February.

Sample Input

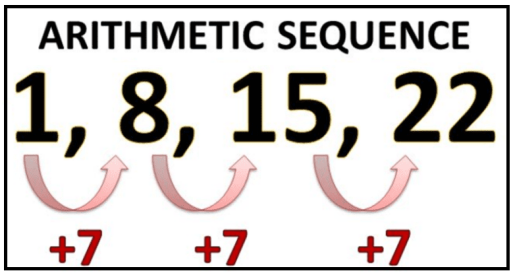
```
5 January
15 December
29 February
11 September
0 #
```

Sample Output

```
You have had your birthday.
Your birthday is still to come.
Sorry, leapling, no birthday this year.
Happy birthday!
```

New Zealand Programming Contest 2021

PROBLEM C **SEQUENCES** **3 POINTS**



An arithmetic sequence is one in which there is some first number, and then a series of numbers which are all a fixed number different.

For example 3, 5, 7, 9 is an arithmetic sequence that has a first number 3. Then each term after that in the sequence is formed by adding 2 to the previous term. (The terms are different by 2). The 3 is also called the first term (term 1) and 9 is the 4th term.

Given a starting number, a difference and a value, your program is to work out if the number could be part of the sequence. If so, output which term that number would be, and if not, output that it is not in the sequence.

Input

Input will consist of a number of lines, where each line has 3 numbers separated by spaces.

The first number is an integer that is the first term in the sequence. The second is the difference - this will be a non-zero integer. The third is the value that you will need to test to determine whether it can be part of the sequence or not. It will not be the same as the first term.

Input is terminated by a zero value for each of the 3 numbers.

Output

Output will consist of one line for each input line. It will consist of either "`Term X`" where X is an integer indicating which term it is, or "`Not in sequence`" if the number isn't part of the sequence

Sample Input

```
3 2 11

-1 -3 -8

0 0 0
```

Output for Sample Input

```
Term 5

Not in sequence
```

Explanation

11 is the 5th term

The sequence is –1, -4, -7, -10

(-1+ -3 = -4, -4 + –3 = -7, -7+ -3 = -10)

-8 isn't in the sequence

New Zealand Programming Contest 2021

PROBLEM D **CAR PARK** **3 POINTS**

Our local car park needs your help. They have a fixed number of spaces for parking and want to be able to tell drivers who are approaching the park just how many spaces are available to use. They also need to stop the entry barrier from opening if the park is full. If your system works, they will sell it to other car park owners.

Input

Input consits of a single scenario representing a single car park. The first line consists of two integers, `S` and `C` (`10 <= S <= 500, 0 <= C <= S`). `S` is the number of spaces in the car park overall, and `C` is the number of cars currently parked there.

The second line consists of a string of from 0 to 255 characters, each character being one of the upper case letters '`I`' or '`O`'. This string represents a stream of data from the entry and exit barriers. `I` means that a car has attempted to come in to the car park, `O` that a car has driven out. If the car park is not full, a car attempting to enter will be allowed in and counted. If all the spaces in the car park are filled with cars, a car attempting to enter will be refused entry and not counted. (Such a car may try again later).

If there are no cars in the car park, an '`O`' in the data stream at that point represents an error, and processing of that scenario should stop.

Output

Output consists of a single line. If there has been an error in the data stream, it will contain just the word "`Error`.". Otherwise, it will show the number of cars in the car park at the end of the scenario in the format "`X cars`." The line ends with a full stop.

Sample Input

```
50 12
IIOIOOIOIIIOOIOIOOII
```

Output for Sample Input

```
14 cars.
```

New Zealand Programming Contest 2021

Problem E **WORD EXTRACTION** **10 points**

I am trying to create a dictionary of all the words in common use by my students. To do this, I am planning to feed text from their work, discussion forum entries and emails through a program that extracts the words.

To make it easy to check if a word has been seen before, I am formatting the text carefully. Firstly, I put everything in lower case, then I strip out all punctuation leaving the words separated by single spaces. If the punctuation comes within a word, it is removed and not replaced with a space. If a "word" consists only of digits, it is ignored. Finally, I sort the words into alphabetical order, removing duplicates.

Punctuation is considered to be anything that is not a letter, a number or white space.

Input

Input will consist of a piece of text to be analysed. It will be a single line which will contain no more than 250 characters, and will contain at least one word which can be extracted.

Output

Output will be the qualifying words extracted from the text, one per line, in alphabetical order.

Sample input (single line which is wrapped round here)

```
Can I please have the spec for the Programming 3 assignment?  B.T.W.
I haven't got the lecture notes either! Sorry, thank you.
```

Output for Sample Input	**Notes**
assignment	The ? after assignment is ignored.
btw	btw – the full stops have been removed from B.T.W.
can	
either	The ! after either is ignored.
for	
got	
have	
havent	havent – the apostrophe has been removed.
i	
lecture	
notes	
please	
programming	
sorry	The , after sorry is ignored.
spec	
thank	
the	
you	The . after you is ignored.

New Zealand Programming Contest 2021

PROBLEM F **POSTMAN JOE** **10 POINTS**

Postman Joe is a fitness fanatic. He has a single row of 20 houses on his delivery route, and sets himself a task every day that will require him to walk up and down the row several times. A typical task will look something like this:

3 U3 D2 U1 U6 D4 D5 U7 U9 D5 D3 U5 U4 D2 D5 U8

This means that Joe delivers firstly to house 3, moves 3 houses up the street and delivers to house 6, then moves 2 houses down the street and delivers to house 4, and so on. The houses are numbered sequentially from 1 to 20, house 1 being at the bottom of the street.

Not every house will necessarily receive a delivery each day, but Joe's instructions must never take him to the same house twice, nor take him beyond either end of the row of houses.

To solve this problem, you must write a program to help Joe. It must check that Joe has set himself a legal task and, if so, must report those houses which do not receive a delivery.

Input

Input will consist of a single task that Joe has set, on a single line.

The task represents the deliveries for a single day. The line starts with a single integer S (1 <=S <= 20) being the first house to which Joe makes a delivery. This is followed by a sequence of letter-number pairs, each separated by a single space. The letter will be U or D, U meaning go up the street (increasing house number), D go down the street. The number will be a single digit integer, stating the number of houses to move.

Output

Output will consist of a single line. If the instructions for the task are legal (ie no house is visited twice, and Joe is not taken beyond either end of the street), output will be a list of houses that do not receive a delivery that day. The list will be in numerical order with a space between each house number. If all houses receive a delivery, output should be the word **none**.

If the instructions for a task are not legal, output should be the word **illegal**.

Turn over for sample input and output

Sample Input 1

```
3 U3 D2 U1 U6 D4 D5 U7 U9 D5 D3 U5 U4 D2 D5 U8
```

Output For Sample Input 1	**Explanation**
`1 8 14 16`	Joe makes 16 deliveries, missing these 4 houses.

Sample Input 2

```
8 D7 U5 U6 D9 U2 D4 U9 U3 D2 U7 U2
```

Output For Sample Input 2	**Explanation**
`illegal`	The 7th delivery (D4) is to house 1 which received the 2nd delivery.

New Zealand Programming Contest 2021

PROBLEM F **HOSTEL NIGHTS** **10 POINTS**

Students staying in a hostel on the first floor have a reputation for being a bit too noisy. The warder decides to investigate reports of noisiness over the course of 5 nights of a week. Other students are not willing to dob in their floor-mates directly but will help him eliminate some rooms where the students were **not** noisy. They are eliminated through being either odd or even, because they are a multiple of some number n, or because the inhabitant's name starts with a particular letter.

Over the 5 nights the warden manages to form 3 variables from various other students' help. At the end of 5 nights he will be able to work out who the noisiest students are. There is at least one as this floor is notorious.

Input

Input gives the data for one week of 5 days. The data for the week begins with 20 lines, each showing a room number and the name of a student, separated by a space. Rooms are numbered from 101 to 120. A student is represented by a single name which begins with an upper case letter.

This names list is followed by 5 lines each representing one night of the week. Each line consists of a letter, a number and another letter, each of which is separated by a space. The first letter is E or O to indicate whether even (E) or odd (O) numbered rooms may be eliminated. The number is used to eliminate all rooms that are a multiple of that number. The second letter may be used to eliminate any students whose name begins with that letter.

Output

Output for each week consists of a list of the noisiest students; that is those who have not been eliminated on the greatest number of nights. Names are output in room number order, one per line.

Turn over for sample input and output.

Sample Input

```
101 Fred
102 Gregory
103 Susan
104 Rewi
105 Albert
106 Georgina
107 Peter
108 Bethany
109 Sarah
110 Justine
111 Barry
112 Matthew
113 Francis
114 Chris
115 Devina
116 Yong
117 William
118 Edward
119 Ruth
120 Luckylast
O 5 G
E 3 F
E 5 S
E 1 A
O 4 C
```

Output for Sample Input

```
Peter
Edward
Ruth
```

Explanation

Noisy students (those not eliminated) on each night are:

Night 1: Rewi, Bethany, Matthew, Chris, Yong, **Edward**

Night 2: Susan, **Peter**, Sarah, Devina, **Ruth**

Night 3: Fred, **Peter**, Barry, Francis, William, **Ruth**

Night 4: (none)

Night 5: Gregory, Georgina, Justine, **Edward**

Peter, Edward and Ruth appear twice each so are the noisiest students.

New Zealand Programming Contest 2021

PROBLEM H **SET GAME** **10 POINTS**

Set is a card game played with a pack of 81 unique cards. Each card in the pack has 4 properties, having one of the three possible values for each property.

Colour – Red (R), Green (G), Blue (B)	**Number** 1, 2, 3,
Shape – Diamond (D), Oval (O), Squiggle (S)	**Fill** – Filled (F), Shaded (S), Empty (E)

This sample contains 3 cards and illustrates all the variations. If this is not in colour, note that the left card is red, the middle is green and the right card is blue.

		
RD1F	GO2S	BS3E

Below each card is its representation, being the value for its colour, shape, number and fill in that order.

The 3 cards illustrated form a set. A set is a collection of 3 cards such that for each property all 3 cards are either the same or different. The 3 cards above are different for all 4 properties. Note that as each card is unique, cards must be different for at least one property.

Here are some other examples.

RD1F	RD1S	RD1E	SET. Same colour, shape and number, different fill.
RD1F	RD1S	RD2E	NOT a set. Two are 1s but one is not.
RD1F	GO2F	BS3E	NOT a set. Two are filled but one is not.
BS3E	BD1E	BO2E	SET. Same colour and fill, different number and shape.

In this problem you will be given data for groups of 3 cards. You must decide whether or not the 3 cards in each group form a set.

Input

Input begins with a single integer, N, on a line by itself (0 < N <= 100). There then follow N lines, each line representing a group of 3 cards. Each card will be represented by 4 characters as described above. The cards will each be separated by a space.

Output

For each line of input, produce one line of output containing either the word "`Set`" or the phrase "`Not a set`".

Sample Input

```
4
RD1F RD1S RD1E
RD1F RD1S RD2E
RD1F GO2F BS3E
BS3E BD1E BO2E
```

Output for Sample Input

```
Set
Not a set
Not a set
Set
```

New Zealand Programming Contest 2021

Problem I **Family Trees** **30 points**

Nowadays New Zealanders come from a variety of backgrounds. This means that any given person's ancestry could include significant contributions from Maori, the Pacific Islands, various western European nations and quite possibly some south east Asian as well. Given a suitable database of family lineages, it should be reasonably simple to determine the proportions of these contributions for any given individual. However, the world has never been simple, so really all we can determine, for any given pair of individuals, is how much the earlier one contributed to the later one (if any). Your task is to write a program to do this.

Input

Input will consist of a series of family trees and queries over that family tree. A family tree consists of a series of lines terminated by a line consisting of a single '#'. Each line contains three different names (each containing fewer than 20 letters) representing the person and his or her parents. Any name will appear first at most once (i.e. everyone has exactly zero or two listed parents) and there are no cycles (i.e. no-one is their own ancestor). The family tree is followed by a series of queries terminated by a line consisting of a single '#'. Each query is a pair of names that may or may not have appeared in the family tree. The file is terminated by a line containing a single '#'.

Output

Output consists of one line for each query in the input. If either of the people named in the query is a direct descendant of the other then print the name of the descendant, a space, the word "`is`", another space, the fraction of ancestry in simplest terms (as shown below), another space, and then the name of the ancestor followed by a full stop ('`.`'). If this is not the case (or if either or both do not appear in the family tree) then the line consists of: "`<`*`name1`*`> and <`*`name2`*`> are not related.`". Leave a blank line between successive family trees.

Turn over for sample input and output.

Sample Input

```
PebblesFlintstone WilmaFlintstone FredFlintstone
LisaRubble PebblesFlintstone BambamRubble
Don Jane FredFlintstone
Paul LisaRubble Don
#
Paul FredFlintstone
FredFlintstone BambamRubble
Jane Don
#
PebblesFlintstone WilmaFlintstone FredFlintstone
LisaRubble PebblesFlintstone BambamRubble
Don Jane FredFlintstone
Paul LisaRubble Don
#
Paul Jane
Jane Tarzan
#
#
```

Output for Sample Input

```
Paul is 3/8 FredFlintstone.
FredFlintstone and BambamRubble are not related.
Don is 1/2 Jane.

Paul is 1/4 Jane.
Jane and Tarzan are not related.
```

PROBLEM J **CHEMISTRY 101** **30 POINTS**

Chemistry is all about reactions—you throw a bunch of stuff into a test-tube, heat it up, hoping that it will neither explode or poison you, then cool it down and try to work out whether what you have is what you expected. That's the easy (and fun) bit—much harder is recording it all. As is usual with skills of this type, chemistry instructors the world over rely on drill—a seemingly endless set of reactions that the students have to complete. The trick is that everything that appears on the left side (the reagents, or 'input') must appear on the right side (the products, or 'output'). This ought to be simple, but generations of chemistry students have demonstrated otherwise.

Professor Plumbius is getting tired of writing the same comments on his student's worksheets over and over and he wants to automate the process. He wants to be able to enter the equations as written by the students and have the computer produce the comments automatically, thus giving him more time to dream up more equations to give his students to practise on. This is where you come in.

Write a program that will read in a chemistry equation and determine whether it is balanced. If it isn't, your program must tell the student what elements are out of balance and by how much.

Normally a chemistry equation is written like this:

$$2H_2O + SO_2 \longleftrightarrow H_2 + H_2SO_4$$

but due to the limitations of computer input we will present it like this:

```
2H2O + SO2 = H2 + H2SO4
```

This example shows the essentials of an equation: each side consists of one or more molecules, separated by '+' signs (the spaces are optional). Each molecule may have a multiplier before it which specifies how many instances of that molecule take part in the reaction. A molecule consists of one or more elements. Each element has a symbol, which is either an uppercase letter, e.g. 'H', or an uppercase letter followed by a lowercase letter, e.g. 'Br'. A symbol may be followed by a multiplier specifying how many atoms of that element are present in that part of the molecule. Thus the first term says that there are two instances of a molecule consisting of two atoms of H and one atom of O. (This happens to be water, but you do not need to know that.)

Given that these are exercises handed out to the students, the left hand sides are, by definition, correct. Thus your job is to determine whether all the atoms that appear on the left also appear on the right. If they do, then the equation is balanced. If not, you must report which elements have been created or lost and how much of each.

Continues on the next page.

Input

Input will consist of a number of equations, each on a line by itself. Each line will contain no more than 250 characters. Each equation represents a set of reagents and a set of products, separated by an '=' sign. Each set will consist of one or more molecules, possibly with multipliers, separated by '+' signs. There may be zero or more spaces on either side of the '+' and '=' signs.

The last line of input will be a '#' on a line by itself. This line should not be processed.

Output

There will be at least one line of output for each equation in the input. If the equation is balanced this line will say '`Equation` *n* `is balanced.`' where *n* is the equation number (starting from 1). If the equation does not balance, then the output line will say '`Equation` *n* `is unbalanced.`' and will be followed by a series of lines of the form

`You have ⟨created` *or* `destroyed⟩` *m* `⟨atom` *or* `atoms⟩ of` *element*`.`

where *element* is the symbol of the element concerned, *m* is the number of extra or missing atoms of that element, and '`atom(s)`' is singular or plural as appropriate. For each unbalanced equation, these lines should be ordered alphabetically by element symbol and terminated by a blank line. (Note that this means that the final line of your output may be a blank line.)

Sample Input

```
2HBr + H2O+SO2=H2+H2SO3+He
2H2O + SO2 = H2 + H2SO4
2H2O+SO2=H2+H2SO4
#
```

Output for Sample Input

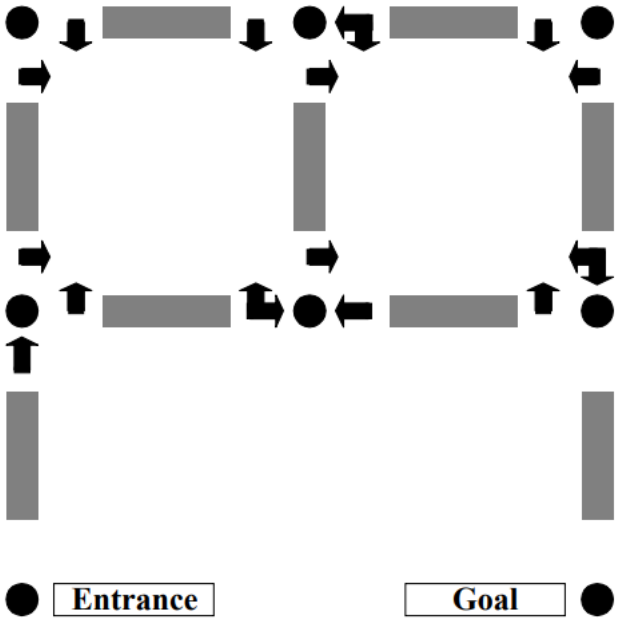
```
Equation 1 is unbalanced.
You have destroyed 2 atoms of Br.
You have created 1 atom of He.

Equation 2 is balanced.
Equation 3 is balanced.
```

PROBLEM K **YET ANOTHER AMAZING MAZE PROBLEM** **30 POINTS**

This maze problem is based on an amazing real maze (YAAM) constructed for people to explore on foot. It turned out to be quite a difficult maze to solve in person, so you have been asked to write a computer program to check to see if mazes of this style can be solved.

As in most mazes, a YAAM maze is traversed by moving from intersection to intersection from a starting location, until the goal intersection is reached. In YAAM mazes however, as each intersection is approached from a given direction, a sign near the entry to the intersection indicates in which directions the intersection can be exited. These directions are always left, forward or right, or any combination of these. A valid solution to the maze must obey these direction signs. The diagram below illustrates a YAAM maze.



The intersections are identified as “(row, column)” pairs, with the upper left being (1,1). The “Entrance” intersection for the given maze is (3,1), and the “Goal” intersection is (3,3). You begin the maze by moving north from (3,1). As you walk from (3,1) to (2,1), the sign at (2,1) indicates that as you approach (2,1) from the south (travelling north) you may continue to go only forward. Continuing forward takes you toward (1,1). The sign at (1,1) as you approach from the south indicates that you may exit (1,1) only by making a right turn. This turns you to the east, walking from (1,1) toward (1,2). So far there have been no choices to be made. This is also the case as you continue to move from (1,2) to (2,2) to (2,3) to (1,3). Next however, after moving west from (1,3) toward (1,2), you have the option of continuing straight or turning left. Continuing straight would take you on toward (1,1), while turning left would take you south to (2,2). The actual (unique) solution to this maze is the following sequence of intersections: (3,1) (2,1) (1,1) (1,2) (2,2) (2,3) (1,3) (1,2) (1,1) (2,1) (2,2) (1,2) (1,3) (2,3) (3,3).

You must write a program to solve valid YAAM arrow mazes. Solving a maze means (if possible) finding a route through the maze that leaves the Entrance in the prescribed direction, and ends in the Goal. This route should not be longer than necessary, of course.

Continues on the next page.

Input

The input will consist of one or more YAAM mazes. The first line of each maze description contains the name of the maze, which is an alphanumeric string of no more than 30 characters. The next line contains, in the following order, the starting row, the starting column, the starting direction, the goal row, and finally the goal column. All are delimited by single spaces. The maximum dimensions of a maze for this problem are 9 by 9, so all row and column numbers are single digits from 1 to 9. The starting direction is one of the characters N, S, E or W, indicating north, south, east and west, respectively.

All remaining input lines for a maze have the format: two integers, one or more groups of characters, and a sentinel asterisk, again all delimited by single spaces. The integers represent the row and column, respectively, of a maze intersection. Each character group represents a sign at that intersection. The first character in the group is N, S, E or W to indicate in what direction of travel the sign would be seen. For example, S indicates that this is the sign that is seen when travelling south. (This is the sign posted at the north entrance to the intersection.) Following this first direction character are one to three arrow characters. These can be L, F or R indicating left, forward, and right, respectively.

The list of intersections is concluded by a line containing a single zero in the first column. The next line of the input starts the next maze, and so on. The end of input is the word END on a single line by itself.

Notes

- The “Entrance” and “Goal” intersections are distinct.
- The “Entrance” and “Goal” intersections have no directions marked on them.
- There is no way of reaching the “Entrance” again once it has been left.

Output

For each maze, the output should contain a line with the name of the maze, followed by a line with either the number of ‘steps’ (edges traversed) of a minimum solution to the maze in the form “`Solved in x steps`” or the phrase “`No Solution Possible`”.

Sample Input

```
SAMPLE
3 1 N 3 3
1 1 WL NR *
1 2 WLF NR ER *
1 3 NL ER *
2 1 SL WR NF *
2 2 SL WF ELF *
2 3 SFR EL *
0
NOSOLUTION
3 1 N 3 2
1 1 WL NR *
1 2 NL ER *
2 1 SL WR NFR *
2 2 SR EL *
0
END
```

Output for Sample Input

```
SAMPLE
Solved in 14 steps
NOSOLUTION
No Solution Possible
```

Explanation

The first maze in the sample input is the maze in the diagram on the previous page.

New Zealand Programming Contest 2021

Problem M **I'm a Frayed Knot** **100 points**

On the table in front of you lie a bunch of pieces of coloured thread. All the threads are different colours and the ends have been arranged in a single line, for example red, blue, green, green, blue, red (see Figure 1). Note that each colour appears exactly twice, once for each end of that thread. You've been asked to tie the threads into a single large loop by successively tying together some pair of adjacent thread ends. In this example you could start by tying end 1 to end 2 (result shown in Figure 2) or end 2 to end 3, but not end 3 to end 4 since that would make a green loop. Likewise, if your first tie was a red to a blue, then your second tie could not join the remaining red and blue.

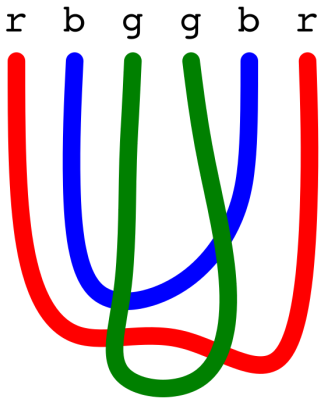


Figure 1

Finding the job a little boring, you decide instead to count the number of ways in which you could perform the task, i.e. the number of sequences of ties that you could do. For instance, suppose that the initial pattern was `rrbb`, then there is only one allowed sequence of ties (join the middle two, then join the two remaining). On the other hand if it were `rbrb`, then there are three allowed sequences (join any consecutive pair to begin with, then join the remaining two).

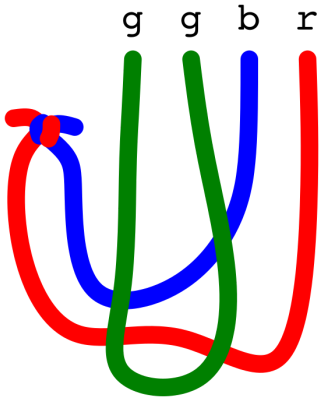


Figure 2

Likewise you can see that if the initial pattern were `rgrbgb`, then four of your initial ties lead to a subsequent pattern of the general form `abab`, while one leads to `abba`. Thus there are 4×3 + 1×2 = 14 ways to complete the ties in this case.

Input

Input will consist of a sequence of colour patterns represented by words of length at most 22 consisting of lower case letters and will be terminated by a line containing the single character '#'.

Output

Output will be a sequence of lines, one for each line in the input, each containing the number of ways to tie off the pattern in the corresponding input line. If an input line is invalid (because it does not contain exactly two occurrences of each of its characters) the output for that line should be 0.

Sample Input

```
rrbb
rbrb
rgrbgb
gbg
#
```

Output for Sample Input

```
1
3
14
0
```

New Zealand Programming Contest 2021

Problem N **Adventure Game** **100 points**

Part of an adventure game requires the adventurer to make her way through a magical maze of numbered rooms. Each room has a set of numbered doors that lead to the indicated rooms. Additionally, each room might contain either a leprechaun or a troll.

Whenever the adventurer visits a room containing a leprechaun, the leprechaun will ensure that she leaves with at least a certain number of gold pieces (GP) in her sack. That is, if she arrives with fewer GP than the prescribed amount, then the leprechaun will "top her up" to that amount; however, if she arrives with the prescribed amount or more, then her supply of GP will remain unchanged. Each time the adventurer visits a room containing a troll, the troll will demand a toll of a certain number of gold pieces which must be paid before continuing.

The adventurer begins with 0 GP and the problem is to determine whether the adventurer can make her way from room 1 to the highest numbered room. If the highest numbered room contains a troll, the adventurer must also be able to pay the toll on arrival.

Input

Input will consist of a sequence of mazes. Each maze will begin with a line containing an integer specifying the number of rooms in the maze (n, $1 \le n \le 1000$). This will be followed by n lines specifying the rooms in ascending order. Each line will specify the contents of the room (empty (E), a leprechaun (L) or a troll (T)), followed by an integer specifying the number of GP that the leprechaun will top up to, or that the troll will require from the adventurer. (For empty rooms this amount is 0.) The rest of the line consists of a list of one or more numbers in the range $1..n$ and terminated by a zero (0), representing the numbers on the doors of that room. The sequence of mazes is terminated by a line containing a single zero (0).

Output

Output will consist of a sequence of lines, one for each maze — either "`Yes`" or "`No`" indicating whether it is possible to reach the final room from the first one.

Sample Input

```
3
E 0 2 0
L 10 3 0
T 15 1 2 0
4
E 0 2 3 0
L 201 2 3 0
L 10 4 0
T 15 2 3 1 0
0
```

Output for Sample Input

```
No
Yes
```

New Zealand Programming Contest 2021

Problem O **List Calculator** **100 points**

Your job, should you choose to accept it, is to write an interpreter for a small list calculator. The input will contain expressions which create and combine lists of numbers.

Operands in expressions are lists of integers. A number alone represents a list with one item.

The operators (from most tightly bound to least tightly bound) are: *parentheses*, *list slicing*, *unary operators*, *binary operators*, and *list concatenation*.

Parentheses `()` are used to achieve control over the order of operations.

The *unary operators* are `+` and `*`, occurring to the left of an expression. They perform list reduction: they repeatedly remove the first two items of a list, and replace them with the result of the operator being applied (addition or multiplication). For example `+(1:2:4)` takes 1 and 2 off the front of the list and applies "+" to them to get 3, which is placed on the front of the list leaving `+(3:4)`; then again it takes the first two items, removes them, applies `+`, and inserts the result into the front of the list leaving `+7` (recall that a number alone represents a list with one item). Now the list has fewer than two items, so the operator returns the list with just the number 7.

The *binary operators* are `+` and `*` written using infix notation (i.e. between two expressions). They are applied to the first element of both lists, then the second element and so on, and return a list with the results. If one list is shorter than the other, then the shorter list is padded with its last item until both lists are the same length. Thus `(1:2:3)+(4:5)` evaluates to `5:7:8`. Binary `*` has higher precedence than binary `+`. The binary and unary operators will never be applied to empty lists.

The `:` operator is used to *concatenate* two lists. It is written using infix notation and returns a list consisting of the items of the left operand followed by the items of the right operand.

List slicing extracts a sublist and is done with the `[begin:end]` operator applied as a postfix to an expression. Items are 0-indexed and a slice includes the beginning item but not the end item. Thus `(1:2:3:4:5)[1:3]` evaluates to `2:3`. The begin or end may be omitted, and that means all items from the beginning or all items until the end respectively. Begin and end are always integer literals (not expressions).

Input is a sequence of *print* and *assignment* commands. Each command is on a single line.

A *print* command is a line consisting of the word `print`, a space, and an expression. It outputs the result of an expression on a single output line. Lists are displayed with ":" as an item separator (and no spaces). The maximum number of items to be output on a line will be no more than 20.

Assignment is provided with the `=` operator. Variable names are single letters and are case-sensitive. A variable can only be assigned once, i.e. it is a constant. Variable definitions can be recursive, which means lists can be infinite. However, the variables will always be defined so that the next element can be directly calculated when required (as in the sample input).

Input will be terminated by a line consisting of a single '#'.

You may assume that all input is syntactically valid.

Turn over for sample input and output.

Sample Input

```
print +(1:2:4)
print (1:2:3:4:5)[1:3]
print (1:2:3)+(4:5)
N=1:(N+1)
E=2*N
print E[:5]
print +E[:10]
F=1:1:(F+F[1:])
print F[:10]
#
```

Output for Sample Input

```
7
2:3
5:7:8
2:4:6:8:10
110
1:1:2:3:5:8:13:21:34:55
```