Problem A: Calculator Language

Calculator Language (CL) supports assignment, positive and negative integers and simple arithmetic. The allowable characters in a CL statement are thus:

A..Z	variable names
0..9	digits
+	addition operator
−	subtraction operator
*	multiplication operator
/	integer division operator
=	assignment operator
()	brackets
_	negative sign

All operators have the same precedence and are right associative, thus $15-8-3 = 15-(8-3) = 10$. As one would expect, brackets will force the expression within them to be evaluated first. Brackets may be nested arbitrarily deeply. An expression never has two operators next to each other (even if separated by a bracket), an assignment operator is always immediately preceded by a variable and the leftmost operator on a line is always an assignment. For readability, spaces may be freely inserted into an expression, except between a negative sign and a number. A negative sign will not appear before a variable. All variables are initialised to zero (0) and retain their values until changed explicitly.

Write a program that will accept and evaluate expressions written in this language. Each expression occupies one line and contains at least one assignment operator, and maybe more.

Input will consist of a series of lines, each line containing a correct CL expression. No line will be longer than 100 characters. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each line of the input. Each line will consist of a list of the final values of all variables whose value changes as a result of the evaluation of that expression. If more than one variable changes value, they should be listed in alphabetical order, separated by commas. If a variable changes value more than once in an expression, only the final value is output. A variable is said to change value if its value after the expression has been evaluated is different from its value before the expression was evaluated. If no variables change value, then print the message 'No Change'. Follow the format shown below exactly.

Sample input

```
A = B = 4
C = (D = 2)*_2
C = D = 2 * _2
F = C - D
E = D * _10
```

```
Z = 10 / 3
#
```

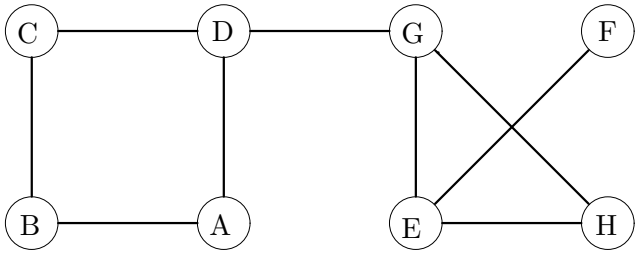
Sample output

```
A = 4, B = 4
C = -4, D = 2
D = -4
No Change
E = 40
Z = 3
```

Problem B: Network Wars

It is the year 2126 and comet Swift-Tuttle has struck the earth as predicted. The resultant explosion emits a large cloud of high energy neutrons that eliminates all human life. The accompanying electro-magnetic storm causes two unusual events: many of the links between various parts of the electronic network are severed, and some postgraduate AI projects begin to merge and mutate, in much the same way as animal life did several million years ago. In a very short time two programs emerge, Paskill and Lisper, which move through the network marking each node they visit: Paskill activates a modified Prolog interpreter and Lisper activates the 'Hello World' program. However 'Hello World' has mutated into an endless loop that so ties up the node that no other program, not even Lisper, can re-enter that node and the Prolog interpreter immediately reverse compiles (and destroys) any program that enters. However, Paskill knows which nodes it has visited and never tries to re-enter them. Thus if Lisper attempts to enter a node already visited by Paskill it will be annihilated; neither can enter a node already visited by Lisper, if either (or both) cannot move both will halt and if they ever arrive at a node simultaneously they annihilate each other. Both programs move through the network at the same speed.

Write a program to simulate these events. All nodes in the the network are labelled with a single uppercase letter as shown below. When moving to the next node, Paskill searches alphabetically forwards from the current node, whereas Lisper searches alphabetically backwards from the current node, both wrapping round if necessary. Thus, (in the absence of the other) if Paskill enters the network below at A, it would visit the nodes in the order A, B, C, D, G, H, E, F; if Lisper enters the network at H it would visit them in the order H, G, E, F. Simulation stops when one or more of the above events occurs. If more than one event occurs, mention Paskill first.



Input will consist of a series of lines. Each line will describe a network and indicate the starting nodes for the two programs. A network is described as a series of nodes separated by ';' and terminated by a period ('.'). Each node is described by its identifier, a ':' and one or more of the nodes connected to it. Each link will be mentioned at least once, as will each node, although not all nodes will be 'described'. After the period will appear the labels of the starting nodes—first Paskill and then Lisper. No line will contain more than 255 characters. The file will be terminated by a line consisting of a single #.

Output will consist of one line for each network. Each line will specify the

terminating event and the node where it occurs. The terminating event is one or two of the following:

- Lisper destroyed in node ?
- {Paskill/Lisper} trapped in node ?
- Both annihilated in node ?

Sample input

```
A:BD;C:BD;F:E;G:DEH;H:EG. A H
E:AB. A B
B:ACD. B D
A:B;B:C;D:E. A D
#
```

Sample output

```
Paskill trapped in node D Lisper trapped in node F
Both annihilated in node E
Lisper destroyed in node B
Lisper trapped in node E
```

Problem C: Withdrawn

For technical reasons, this problem has been withdrawn.

Problem D: Strategy

A well known psychology experiment involves people playing a game in which they can either trade with each other or attempt to cheat the other player. If both players TRADE then each gains one point. If one TRADEs and the other CHEATs then the TRADEr loses 2 points and the CHEATer wins 2. If both CHEAT then each loses 1 point.

There are a variety of different strategies for playing this game, although most people are either unable to find a winning strategy, or, having decided on a strategy, do not stick to it. Thus it is fairer to attempt to evaluate these strategies by simulation on a computer. Each strategy is simulated by an automaton. An automaton is characterised by a program incorporating the strategy, a memory for previous encounters and a count reflecting the score of that automaton. The count starts at zero and is altered according to the above rules after each encounter. The memory is able to determine what happened on up to the last two encounters with each other contender.

Write a program that will read in details of up to 10 different strategies, play each strategy against each other strategy 10 times and then print out the final scores. Strategies will be in the form of simple programs obeying the following grammar:

<program> ::= <statement>.
<statement> ::= <command> | <ifstat>
<ifstat> ::= IF <condition> THEN <statement> ELSE <statement>
<condition> ::= <cond> | <cond> <op> <condition>
<op> ::= AND | OR
<cond> ::= <memory> {= | #} {<command> | NULL}
<memory> ::= {MY | YOUR} LAST {1 | 2}
<command> ::= TRADE | CHEAT

Note that LAST1 refers to the previous encounter between these two automata, LAST2 to the encounter before that and that 'MY' and 'YOUR' have the obvious meanings. Spaces and line breaks may appear anywhere in the program and are for legibility only. The symbol '#' means 'is not equal to'. NULL indicates that an encounter has not ocurred. The following are valid programs:

```
CHEAT.
IF MY LAST1 = CHEAT THEN TRADE ELSE CHEAT.
IFYOURLAST2=NULLTHENTRADEELSEIFYOURLAST1=TRADETHENTRADE
ELSECHEAT.
```

Input will consist of a series of programs. Each program will be no longer than 255 characters and may be split over several lines for convenience. There will be no more than 10 programs. The file will be terminated by a line containing only a single '#'.

Output will consist of one line for each line of input. Each line will consist of the final score of the relevant program right justified in a field of width 3.

Sample input

```
CHEAT.
IF MY LAST1 = CHEAT THEN TRADE ELSE CHEAT.
IFYOURLAST2=NULLTHENTRADEELSEIFYOURLAST1=TRADETHENTRADE
ELSECHEAT.
#
```

Sample output

```
  1
-12
-13
```

Problem E: Keywords

Many researchers are faced with an ever increasing number of journal articles to read and find it difficult to locate papers of relevance to their particular lines of research. However, it is possible to subscribe to various services which claim that they will find articles that fit an 'interest profile' that you supply, and pass them on to you. One simple way of performing such a search is to determine whether a pair of keywords occurs 'sufficiently' close to each other in the title of an article. The threshold is determined by the researchers themselves, and refers to the number of words that may occur between the pair of keywords. Thus an archeologist interested in cave paintings could specify her profile as "0 rock art", meaning that she wants all titles in which the words "rock" and "art" appear with 0 words in between, that is next to each other. This would select not only "Rock Art of the Maori" but also "Pop Art, Rock, and the Art of Hang-glider Maintenance".

Write a program that will read in a series of profiles followed by a series of titles and determine which of the titles (if any) are selected by each of the profiles. A title is selected by a profile if at least one pair of keywords from the profile is found in the title, separated by no more than the given threshold. For the purposes of this program, a word is a sequence of letters, preceded by one or more blanks and terminated by a blank or the end of line marker.

Input will consist of no more than 50 profiles followed by no more than 250 titles. Each profile and title will be numbered in the order of their appearance, starting from 1, although the numbers will not appear in the file. Each profile will start with the characters "P:", and will consist of a number representing a threshold, followed by two or more keywords in lower case. Each title will start with the characters "T:", and will consist of a string of characters terminated by "|". The character "|" will not occur anywhere in a title except at the end. No title will be longer than 255 characters, and if necessary it will flow on to more than one line. No line will be longer than eighty characters and each continuation line of a title will start with at least one blank. Line breaks will only occur between words. All non-alphabetic characters are to be ignored, thus the title "Don't Rock — the Boat as Metaphor in 1984" would be treated as "Dont Rock the Boat as Metaphor in" and "HP2100X" will be treated as "HPX". The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each profile in the input. Each line will consist of the profile number (the number of its appearance in the input) followed by ":" and the numbers of the selected titles in numerical order, separated by commas and with no spaces.

Sample input

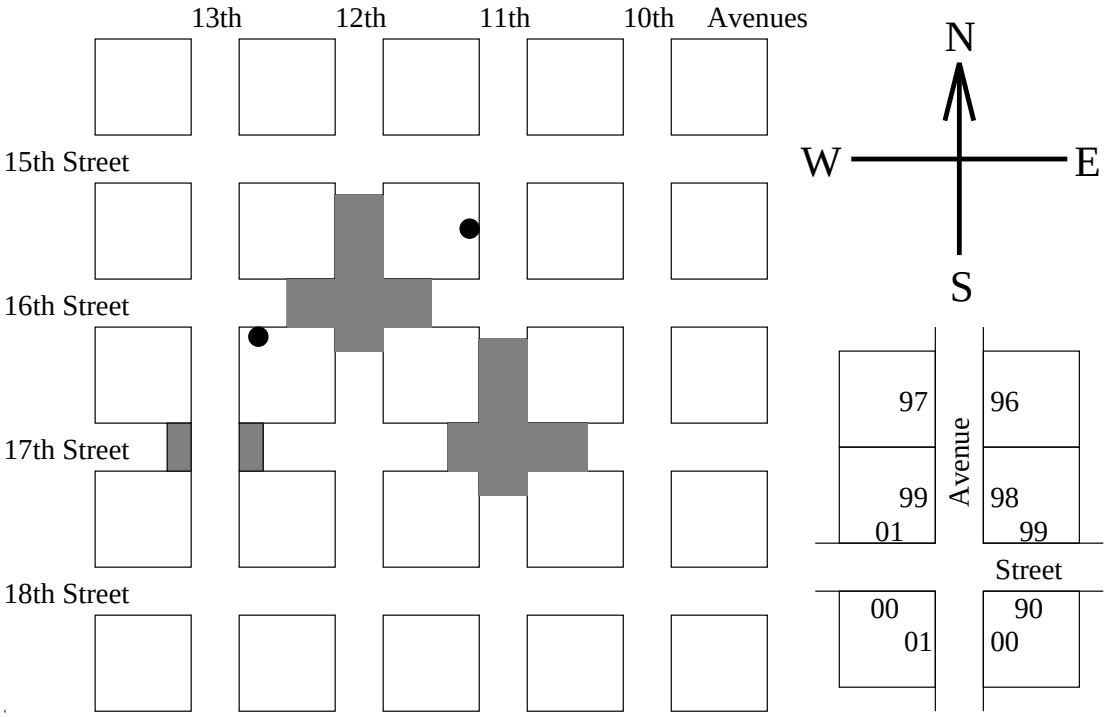
```
P: 0 rock art
P: 3 concepts conceptions
P: 1   art rock   metaphor concepts
T: Rock Art of the Maori|
T: Jazz and Rock - Art Brubeck and Elvis Presley|
T: Don't Rock --- the Boat as Metaphor in 1984, Concepts
   and (Mis)-Conceptions of an Art Historian.|
T: Carved in Rock, The Art and Craft of making promises
   believable when your 'phone bills have gone
   through the roof|
#
```

Sample output

```
1: 1,2
2:
3: 1,2,3,4
```

Problem F: City Navigation

Most US cities are constructed according to a very simple plan—they have Avenues running north and south, and Streets running east and west, enclosing square blocks. Avenues and Streets are numbered, with numbers increasing westward and southward. There are 50 driveways on each side of a block, numbered 00 to 98 on one side and 01 to 99 on the other. House numbers increase in the same directions as Street and Avenue numbers. If you are travelling in the direction of increasing numbers then odd numbers are on your right. Thus the house at 1288 16th Street (S16 1288) is located on 16th Street, west of 12th Avenue and east of 13th Avenue, and is on the right-hand side going east. The residence described as A11 1543 lies on 11th Avenue, south of 15th Street and north of 16th Street, and is on the right-hand side going south. Both of these are marked on the following typical street map:



Quiet suburbs are formed by the simple expedient of making some Avenues and Streets discontinuous as shown above. Note that Avenues and Streets keep the same name, even when there are places where they simply don't exist. It is difficult to get lost in such a city, as the address tells you exactly where to go. However, if you don't know the pattern of missing portions, you can spend a lot of time going into dead-end roads.

Write a program that will firstly read in a description of the 'missing' areas in a city and then a series of pairs of addresses, where an address is assumed to specify a driveway not necessarily a residence. For each pair of addresses the program must calculate the distance between them, by the shortest legal route.

The distance is the number of driveways you pass (on your side of the road) excluding the source and destination. You may make the following assumptions:

- You drive on the right hand side of the road.
- You may not cross a lane of traffic except at an intersection, that is you must turn right when entering or leaving a driveway.
- Driveways are located in the centres of their sections.
- U-turns are illegal except at the end of cul de sacs.
- Streets and Avenues are numbered from 00 to 49 and there are no roads beyond these bounds; however there are driveways on both sides of the bounding roads.
- Sections on corners have two driveways.
- A route exists between any pair of driveways.

Input will be divided into two portions: a "missing road" portion and an address portion, each terminated by a line consisting of a single '#'. The "missing road" portion consists of a series of lines with each line containing a road identifier and a pair of house numbers. A road identifier is an 'A' or an 'S' (specifying an Avenue or a Street) followed by a number in the range 00 to 49. A house number is an even number in the range 0000 to 4898. The area between and including the specified numbers on the identified road is inaccessible. Note that the line goes directly across the street, thus if number 1612 is inaccessible, then so is 1613. Inaccessible portions run from the borders of sections not from driveways. There will be exactly one space separating parts of the input.

The address portion consists of a series of lines each line containing two addresses. An address is a road identifier (as above) followed by a number in the range 0000 to 4899. There will be exactly one space separating parts of the input.

Output consists of a series of lines, one for each line in the address portion of the input file. Each line contains the distance between the two houses specified in the input (the number of driveways passed) written as an integer, left justified.

The following sample data matches the diagram on previous page. (Note the intersection of A13 and S17).

Sample input

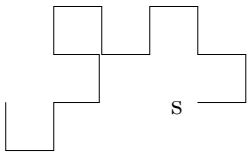
```
A11 1612 1720
A12 1508 1636
S16 1152 1250
S17 1048 1134
S17 1272 1326
#
S16 1288 A11 1543
#
```

Sample output

```
213
```

Problem G: Paper Folding

If a large sheet of paper is folded in half, then in half again, etc, with all the folds parallel, then opened up flat, there are a series of parallel creases, some pointing up and some down, dividing the paper into fractions of the original length. If the paper is only opened "half-way" up, so every crease forms a 90 degree angle, then (viewed end-on) it forms a "dragon curve". For example, if four successive folds are made, then the following curve is seen (note that it does not cross itself, but two corners touch):



Write a program to draw the curve which appears after N folds. The exact specification of the curve is as follows: The paper starts flat, with the "start edge" on the left, looking at it from above. The right half is folded over so it lies on top of the left half, then the right half of the new double sheet is folded on top of the left, to form a 4-thick sheet, and so on, for N folds. Then every fold is opened from a 180 degree bend to a 90 degree bend. Finally the bottom edge of the paper is viewed end-on to see the dragon curve. From this view, the only unchanged part of the original paper is the piece containing the "start edge", and this piece will be horizontal, with the "start edge" on the left. This uniquely defines the curve. In the above picture, the "start edge" is the left end of the rightmost bottom horizontal piece (marked 's'). Horizontal pieces are to be displayed with the underscore character, and vertical pieces with the "|" character.

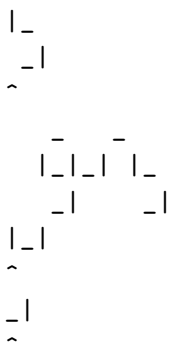
Input will consist of a series of lines, each with a single number N ($1 \leq N \leq 13$). The end of the input will be marked by a line containing a zero.

Output will consist of a series of dragon curves, one for each value of N in the input. Your picture must be shifted as far left, and as high as possible. Note that for large N, the picture will be greater than 80 characters wide, so it will look messy on the screen. The pattern for each different number of folds is terminated by a line containing a single '^'.

Sample input

```
2
4
1
0
```

Sample output



Problem H: Shuffling Patience

Many children enjoy playing cards, especially some of the simpler forms of patience or solitaire, yet many of them find it difficult to shuffle the cards adequately. The following 'patience' game assists this as well as aiding card recognition and boosting simple arithmetic skills. The essence is to 'cover' exposed pairs or triples of cards that have a specific relationship to each other.

A deck of cards consists of 52 cards, in four suits of 13 ranks. The suits are spades, hearts, clubs, diamonds and the ranks run from ace (face value one), 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen and king. During play up to 16 piles in a 4 by 4 grid may be created if necessary, although usually fewer are needed. Deal cards, face up, in this 4x4 grid. Before playing each card check whether a pair or triple of cards already played can be covered. A pair of cards can be covered if they are of rank ace to ten and their face values add to 11. A triple of cards can be covered if they form the set {jack, queen, king}. If no pairs or triples exist, a new pile is started.

Where more than one pair and/or triple exists, only one is covered before reassessing. Cards are always covered in the same order they were dealt, that is left to right, top to bottom. The first card covered shall be the eligible card nearest the start of play. The second card covered (and also the third for a triple) is its partner nearest the start of play. Thus if the first part of a deck consists of: TS QC 8S 8D QH 2D 3H KH 9H..., then the first seven cards will be played as follows:

TS	QC	8S	8D
QH	2D	3H	

The next two cards (KH 9H) will then cover the pair 8S and 3H respectively. Note that covering pairs or triples is considered an indivisible operation, and thus further covering operations are not considered until it is complete.

Write a program to simulate the playing of this game. Your program must read in one or more decks of cards, simulate the play and determine how many cards are on each pile at the end. If it is not possible to remain within the stipulated 16 piles, terminate that deal with a message as described below.

Input will consist of a series of decks of cards, each deck specified as 4 lines each containing 13 cards. Each card will be specified by two characters, a rank (A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K) followed by a suit (S, H, C, D). Cards will be in the order in which they will be played. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one line for each deck in the input. Each line shall start with the deck number, followed by a colon. If it is not possible to play a deck within the specified 4x4 grid, then write a space followed by the message 'Overflowed on card no' followed by the number of the card about to be dealt. If it is possible to play the entire deck, then write out the non-zero numbers that represent the numbers of cards in each pile when the deck is fully dealt. **All** numbers are to be right justified in a field 3 characters wide.

Sample input

```
TS QC 8S 8D QH 2D 3H KH 9H 2H TH KS KC
9D JH 7H JD 2S QS TD 2C 4H 5H AD 4D 5D
6D 4S 9S 5S 7S JS 8H 3D 8C 3S 4C 6S 9C
AS 7C AH 6H KD JC 7D AC 5C TC QD 6C 3C
#
```

Sample output

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