1. Chocolate Toxicity Analyzer

Program Name: cta.java Input File: cta.in

The local vet clinic has asked you to write a program that, given the circumstances of a situation where a dog has consumed chocolate, determines a suggested treatment. You know the following facts:

On average,
Milk chocolate contains 44 mg of theobromine per oz.
Semisweet chocolate contains 150mg of theobromine per oz.
Baker's chocolate contains 390mg of theobromine per oz.

And suggested treatments for the amount of theobromine consumed/dog's body weight are as follows:

< 20mg/kg	Monitor animal's behavior.
20-100mg/kg	Induce vomiting and administer activated charcoal. Animal may return home.
> 100mg/kg	Induce vomiting and administer activated charcoal. Leave animal at clinic.

Input
The first line of the input file will contain a single integer, *m,* indicating the number of data sets.
The next *m* lines will each contain "*ChocolateType ChocolateAmount DogWeight*", where:

1. "*ChocolateType*" will be one of the following: "Milk", "Semisweet", or "Baker's".
2. "*ChocolateAmount*" will be an integer (1-32) indicating the amount of chocolate in oz consumed.
3. "*DogWeight*" will be an integer (5-150) indicating the weight of the dog in kg.

Output
For each dataset, output a single line containing the suggested treatment given the parameters and above table.

Example Input File
```
3
Milk 10 20
Semisweet 30 40
Baker's 1 100
```

Example Output To Screen
```
Induce vomiting and administer activated charcoal. Animal may return home.
Induce vomiting and administer activated charcoal. Leave animal at clinic.
Monitor animal's behavior.
```

2. Go the Distance

Program Name: distance.java Input File: distance.in

Given a sequence of unsorted numbers, determine how badly out of order they are.

Write a program that, for any given sequence of unique natural numbers, will compute the 'distance' between that original ordering and the same set of numbers sorted in ascending order. The distance should be computed by calculating how far displaced each number is in the original ordering from its correct location in the sorted list and summing those results.

For instance, given the list 9 2 5 6 3, the target list would be 2 3 5 6 9. In this case, the 9 is four positions out of place, the 2 is one position out of place, the 5 and 6 are in the correct locations, and the 3 is three positions out of place. Therefore, the distance is 4+1+0+0+3=8.

Input

The first line of input will consist of a single integer, *n*, indicating the number of lines in the input. Each of the following *n* lines represents one sequence of numbers requiring analysis. These lines all begin with an additional integer, *m*, indicating the number of elements in the sequence.

Lists will contain at most 20 elements, each of which will be less than 100.

Output

For each sequence in the input, display the distance between the given ordering and and its sorted (ascending) counterpart.

Example Input File

```
3
5 9 2 5 6 3
20 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
3 1 49 99
```

Example Output To Screen

```
8
38
0
```

3. All That Jazz

Program Name: jazz.java Input File: jazz.in

Benny is a struggling musician trying to determine if the local nightclub he works for is stealing his music. Help Benny compare samples of each melody.

Input

The first line of input will consist of a single integer, *n*, indicating the number of data sets in the input. The remainder of the input consists of those *n* data sets.

Each data set will consist of two lines with the first line containing Benny's melody and the second line containing the nightclub's melody. Each melody will be a sequence of notes, with each note represented by a single character A-G.

Melodies are considered matching if the notes in Benny's melody exist in the nightclub's melody in the same order. For example:
ABCD
would match
AGBGCGDG
since ABCD exists in the second melody (with added G notes)
but
ABCD
would not match
AGBGDGCG
since ABCD does not exist (in that order) in the second melody.

Output

For each dataset in the input, display the line, "There is no match." if the melodies do not match or "There is a match consisting of *X* notes.", if the melodies match, where *X* is the length of the shortest sequence of notes that match.

Example Input File

```
4
ABCD
AGBGCGDG
ABCD
AGBGDGCG
CEFGCEFGCEFGECED
ABABABABABABABACEFBGCAEBFGCAEFGECBEADED
ABCD
ABCAAADABCAADABCADABCDA
```

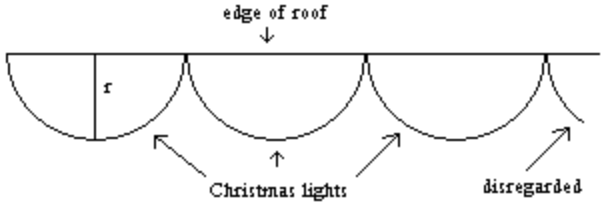
Example Output To Screen

```
There is a match consisting of 7 notes.
There is no match.
There is a match consisting of 22 notes.
There is a match consisting of 4 notes.
```

4. Christmas Lights

Program Name: lights.java Input File: lights.in

You have been tasked with hanging up your neighbor's Christmas lights. Your neighbor wants the lights hung in a series of half-circles from the edge of his roof:



Given the length of the string of Christmas lights and the radius of the series, determine how long the edge of a roof you will be able to decorate. Note your neighbor only wants complete half-circles in his decorations, so disregard any extra length of Christmas lights that do not form a complete half-circle.

Input

The first line of input will contain a single integer, *n*, indicating the number of data sets to process. The remainder of the input consists of those *n* data sets.

Each data set will consist of a single line containing "*r x*", where:

1. *r* is an integer (1-10000) describing the radius for the series.
2. *x* is an integer (1-10000) describing the length of the string of Christmas lights.

Note that all measurements given are in centimeters.

Output

For each data set in the input display the length (in centimeters) of the edge of the roof you are able to decorate.

Example Input File

```
2
30 3000
30 3016
```

Example Output To Screen

```
1860
1920
```

5. Parse This!

Program Name: parse.java Input File: parse.in

Parse, parse, parse. You love to parse. You want to take a group of words and determine if they form a sentence. For the purposes of this problem, a sentence <S> can be defined in terms of its parts (noun phrases <NP>, verb phrases <VP>, prepositional phrases <PP>, and other parts of speech) using the following rules:

```
<S> = <NP> <VP> [ <conjunction> <NP> <VP> ]
<NP> = [ <article> ] [ <adjective> ] <noun>
<VP> = <verb> [ <adverb> ] [ <PP> ]
<PP> = <preposition> <NP>
<noun> = I | bridge | boy | dog | pizza | home | ball | store
<verb> = threw | ran | bought | eats | buy | loves | went
<adjective> = big | tall | tasty | round | blue
<adverb> = quickly | loudly
<preposition> = to | under | from
<article> = a | an | the
<conjunction> = and | or | but | yet
```

Angle brackets indicate a placeholder that has its own rule defining it.
Square brackets indicate optional elements.
Vertical lines separate a list of choices, exactly one of which must be matched.
Note that the rules are case-insensitive ("the" is equivalent to "The").

Input

The first line of input will consist of a single integer, *n*, indicating the number of datasets in the remaining input. Each dataset will consist of:

1. A single line consisting solely of words and spaces (no punctuation), where a word is a contiguous unit of alphabetic characters.

Output

For each dataset in the input, output a single line with the phrase "Sentence" if the given line is a sentence according to the described rules. Otherwise, output a single line with the phrase "Not a sentence".

Example Input File

```
4
I went quickly to the big store and I bought
The dog eats quickly
George eats pizza
Tim eats and loves and threw
```

Example Output To Screen

```
Sentence
Sentence
Not a sentence
Not a sentence
```

6. Pegged

Program Name: pegged.java Input File: pegged.in

Write a program that, given a peg game starting position and a limited number of moves, can determine the best achievable score.

Here is the peg board layout along with the number of points scored for a peg in each position:

```
  ...       999
  ...       444
.......   9411149
.......   9410149
.......   9411149
  ...       444
  ...       999
```

Lower scores are better, with the best possible score being 0 (one remaining peg in the central hole).

A single move consists of 'jumping' one peg horizontally or vertically over an adjacent peg into an empty hole and then removing the jumped peg.

Input

The first line of input will consist of a single integer, *n*, indicating the number of game boards to analyze. The remainder of the input consists of those *n* game boards. Each hole on the game board will either be empty or have a peg in it. An empty hole will be represented by a period while a peg will be represented by the lower-case letter 'o'.

Output

For each board in the input, output a single line with the statement, "The best score for board #X is Y points." The value of X will be 1 for the first board, 2 for the second, etc. The value of Y will be the best possible score that can be achieved with a maximum of 10 moves. Note, it is acceptable to use fewer than the maximum number of moves.

Example Input File

```
3
   ...
   ...
.......
....oo.
.......
   ...
   ...
   ...
   ...
..ooo..
..o.o..
..ooo..
   ...
   ...
   ...
   ...
.......
.ooooo.
.......
   ...
   ...
```

Example Output To Screen

```
The best score for board #1 is 0 points.
The best score for board #2 is 2 points.
The best score for board #3 is 10 points.
```

7. Login Prompt

Program Name: prompt.java **Input File: prompt.in**

Write a program that will display a Unix login prompt.

Input

This program requires no input.

Output

Output the line, “login:”

Example Output To Screen

```
login:
```

8. Magazine Quiz

Program Name: quiz.java Input File: quiz.in

You are writing a program to score a quiz for an online magazine. The quiz is multiple-choice and consists of 9 questions, each having four choices (A, B, C, or D) each having a point value of -1, 0, +1, or +2. The point value assigned to a given choice for each question varies by the chart below where answers for each problem are listed such that the first choice listed is worth -1, the second is worth 0, the third, +1, and the fourth, +2:

#	-1	0	+1	+2
1	D	A	B	C
2	C	B	D	A
3	B	D	A	C
4	A	B	C	D
5	A	C	B	D
6	B	D	A	C
7	B	C	D	A
8	A	C	B	D
9	C	B	D	A

Scores for the quiz fall into the following four ranges, each with its own one-line diagnosis:

SCORE	DIAGNOSIS
< 0	"It's hopeless!"
0 to 5	"Don't give up!"
6 to 10	"You're on target!"
> 10	"It's a sure thing!"

Input
The first line of input will contain a single integer, *n*, indicating the number of data sets. Each of the next *n* lines will contain a string of 9 characters representing one person's answers for the quiz.

Output
For each data set, output the score and the diagnosis.

Example Input File
```
4
BDACBADBD
CACDDCADA
DCBAABBAC
ABDBCDCCB
```

Example Output To Screen
```
9 You're on target!
18 It's a sure thing!
-9 It's hopeless!
0 Don't give up!
```

9. Space Race

Program Name: race.java Input File: race.in

Space Race is a board game consisting of 1000 squares with players starting at square 1 and the goal at square 1000. Players take turns rolling a 20-sided die and moving the indicated number of squares ahead on the game board. There are two types of special squares which have instructions that must be followed immediately if the square was arrived at via a die roll. The special squares types are:

Square Type	Special Instructions
prime number	skip ahead to the next prime numbered square
perfect square	move backward to the prior perfect square numbered square (yes, that's grammatically correct)

For example, if a player landed in square 64 (8^2) by a die roll, their piece would be moved back to square 49 (7^2). Similarly, if a player landed in square 5 by a die roll, their piece would be moved forward to square 7.

Input

The first line of input will consist of a single integer, *n*, indicating the number of games to analyze. Each of the next *n* lines represents a player's die rolls, with the first number on the line indicating the number of die rolls contained in the remainder of the line.

Output

For each line of die rolls in the input, display the message "Player #X ended up in square Y". X is 1 for the first player, 2 for the second, etc., and Y is the square the player ended on. Note that a player is moved to square 1000 if he/she lands on it directly, rolls to move beyond it, or the prime number rule attempts to move them beyond it. Once a player has reached square 1000, they cease to move.

Example Input File

```
3
1 1
1 3
3 4 2 1
```

Example Output To Screen

```
Player #1 ended up in square 3
Player #2 ended up in square 1
Player #3 ended up in square 7
```

10. Txtspk

Program Name: text.java Input File: text.in

To save time when sending text messages, many people abbreviate what they are trying to say. One such way is to perform the following steps:

1. Remove all punctuation from the message. For this program, any non-alphanumeric, non-whitespace character is considered punctuation.
2. Convert all uppercase letters to lowercase.
3. Remove all vowels (a,e,i,o,u) from any word (contiguous unit of alphanumeric characters) of four alphanumeric characters or more.

Input

The first line of input will contain a single integer, *n*, indicating the number of data sets to process. The remainder of the input consists of those *n* data sets.

Each data set will consist of a single line containing a message (1-80 characters) to convert.

Output

For each data set in the input display the text message resulting from applying the above steps.

Example Input File

```
3
Tim, do you want to go grab a pizza?
Dear John, I am leaving you.
I will meet you at the theatre at 9!
```

Example Output To Screen

```
tim do you wnt to go grb a pzz
dr jhn i am lvng you
i wll mt you at the thtr at 9
```

11. Tic-Tac-Win!

Program Name: toe.java Input File: toe.in

Given the current state of a tic-tac-toe board, determine the best move. Here is a chart for determining the strategic value (or 'goodness') of a move. Moves listed first are better:

1. The move causes you to win immediately.
2. The move blocks the other player from a sure win on the next move.
3. The move gives you two potential one-move wins (i.e., you are assured the win since only one can be blocked).
4. The move gives you one potential one-move win.
5. The move is in the middle square.
6. The move is in the top-right square.

For those unfamiliar with the game of tic-tac-toe, the game requires two players who take turns making marks on empty squares of a 3x3 grid. The first player makes an 'X' mark, and the second player makes an 'O' mark, and each is attempting to be the first to have three of his or her marks in a line (vertical, diagonal, or horizontal).

Input

The first line of input will consist of a single integer, *n*, indicating the number of game boards in the input. The following 3*n* lines contain the game boards, with X/O marks and periods representing blanks.

Output

For each game board in the input, display the line, "Board #X", where X is 1 for the first data set, 2 for the second, etc. Then display the game board with the best next move for the 'O' player marked. Note that there will always be a single best move to choose (i.e., no ties).

Example Input File

```
3
.XO
.X.
...
OXO
OXX
..X
...
.X.
...
```

Example Output To Screen

```
Board #1
.XO
.X.
.O.
Board #2
OXO
OXX
O.X
Board #3
..O
.X.
...
```

12. Witness Protection?

Program Name: witness.java **Input File: witness.in**

The federal witness protection program is cutting costs and needs you to write a program to generate new names for protected witnesses. Names will be based on the birth dates and genders of the witnesses using the following scheme:

Birth Month	Female First Name	Male First Name
January	Janice	Jimmy
February	Fanny	Freddie
March	Mae	Mack
April	April	Anthony
May	Meadow	Manny
June	Joanne	Junior
July	Jackie	Jason
August	Adriana	Artie
September	Sophia	Stoolie
October	Olivia	Otto
November	Nell	Nick
December	Donna	Dino

Birth Day (of month)	Middle Name (both genders)
1-15	Carmine
16+	Corleone

Last Name is always
Smith

Input
The first line of input will consist of a single integer, *n*, indicating the number of witnesses. The next *n* lines will each contain one witness birth date in MM/DD format followed by the witness gender (either 'M' or 'F').

Output
For each birth date in the input, output the witnesses new name on its own line.

Example Input File
```
3
07/21 M
09/29 F
01/01 M
```

Example Output To Screen
```
Jason Corleone Smith
Sophia Corleone Smith
Jimmy Carmine Smith
```