Cherry Blossom Counting (prob1)

The Problem

A little known fact about Macon, GA is that it has more cherry blossoms than any other city in the world! A tour company in town offers various tours and would like to add to its pamphlet how many cherry blossoms visitors will see on each tour. Help the tour company with the calculation!

Input

The first line of the input will contain a single positive integer, N ($N \le 100$), representing the number of locations in Macon that different tours potentially visit. The second line of the input will contain N space separated integers, B_1, B_2, …, B_N, ($0 \le B_i \le 1000000$, for all $1 \le i \le N$) representing the number of cherry blossoms at tour locations 1 through N, in numerical order. The third line of the input will contain a single positive integer, T ($T \le 100$), representing the number of different tours offered by the tour company. The following T lines contain descriptions of each of the tours. Each tour will be given by two space separated integers, S_i and F_i, representing the starting tour location and the ending tour location, respectively. The tour will visit each numbered location in between S_i ($1 \le S_i \le N$)and F_i ($S_i \le F_i \le N$), inclusive.

Output

For each tour, output a single line with the total number of cherry blossoms visitors will see on that tour.

Sample Input	**Sample Output**

8
100 0 1234 1000000 88 912 49110 6
3
1 8
2 2
5 7
	1051450
	0
	50110

Snowpocalypse (prob2)

The Problem

Boston's brutal winter of 2015 has made its way into the record books. After yet another blizzard last week, the city marked its snowiest month since record-keeping started in 1872. Your job is to take different weather reports of snowfall totals in inches from around the city, and convert them into equivalent feet & inches as well as corresponding rainfall amount. Ten inches of snow is equivalent to one inch of rain. Remember also that there are 12 inches in one foot.

Input

The first line of input contains a single integer *n*, ($1 \leq n \leq 1000$) which is the number of test cases which follow. Each test case is a single line starting with a single integer *t* ($1 \leq t \leq 100$) representing inches of snow reported at a different location in Boston.

Output

For each test case, output the equivalent feet and inches followed by the rainfall equivalent in the format below. Use integer division and whole numbers only for all calculations. Always use the plural word inches.

Sample Input

```
3
45
39
57
```

Sample Output

```
3 feet, 9 inches of snow is the equivalent of 4 inches of rain
3 feet, 3 inches of snow is the equivalent of 3 inches of rain
4 feet, 9 inches of snow is the equivalent of 5 inches of rain
```

Fizzy Buzzy Bears (prob3)

The Problem

FizzBuzz is a popular programming exercise that is often used in job interviews (really, see http://en.wikipedia.org/wiki/Fizz_buzz) . But you're at a programming contest at Mercer University, not in a job interview, so instead of the classic FizzBuzz, you'll do Fizzy Buzzy Bears.

To play Fizzy Buzzy Bears, you'll print out a series of integers from a given low to a given high bound, except when the numbers are divisible by special values:

- If the number is divisible by 3 (the number of main Mercer campuses), print "Mercer" instead of the number.
- If the number is divisible by 4 (the number of Mercer Regional Academic Centers), print "Bears" instead of the number.
- If the number is divisible by 18 (Mercer was founded in 1833), print "Grrrr" (there are 4 r's) instead of the number.
- If the number is divisible by 33 (Mercer was founded in 1833), print "Bite" instead of the number.

If a number is divisible by more than 1 of 3, 4, 18, or 33, print all the applicable words in order of the divisors, so 12 should print "MercerBears", 36 should print "MercerBearsGrrrr", and 66 should print "MercerBite".

Input

The input will consist of one or more input sets. Each set will have two integers, a low and high bound. The end of input will be an input set with a low bound that is greater than the high bound. This last case should not be processed.

Output

For each input set, have a line with the case number (where the first case is numbered 1 and the others are numbered sequentially). Then list the integers from the lower bound to the upper bound, inclusive, with the Fizzy Buzzy Bear value if there is one or the integer, one per line.

Have a blank line after the output for each input set.

Sample Input

```
1 10
5 20
30 35
10 1
```

Sample Output

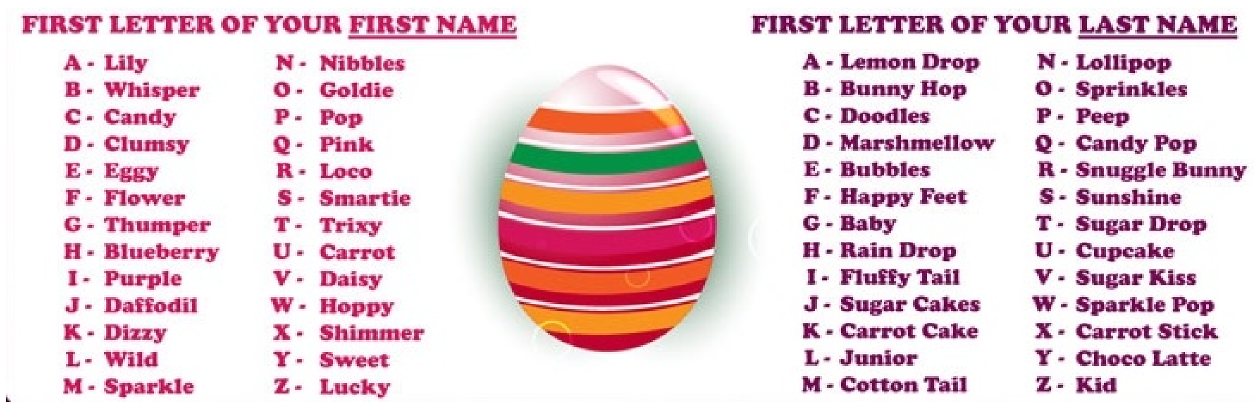
```
Case 1:
1
2
Mercer
Bears
5
Mercer
7
Bears
Mercer
10

Case 2:
5
Mercer
7
Bears
Mercer
10
11
MercerBears
13
14
Mercer
Bears
17
MercerGrrrr
19
Bears

Case 3:
Mercer
31
Bears
MercerBite
34
35
```

Find Your Easter Bunny Name (prob4)

The Problem

Every spring in Macon, GA, the local family life radio station, WJTG 91.3 FM, comes out with a list of “bunny names” for everyone just in time for the Easter season. Your job is to write a program to input this year’s list of names and then convert a series of one or more real names into their corresponding Easter bunny names.

To convert a real name to an Easter bunny name, you take the first letter from the real first name and look up in a table the new bunny first name. You would then take the first letter of the real last name and look up its corresponding bunny last name. Join these together to form the new Easter bunny name. Here is a sample table.



FIRST LETTER OF YOUR FIRST NAME		FIRST LETTER OF YOUR LAST NAME	
A - Lily	N - Nibbles	A - Lemon Drop	N - Lollipop
B - Whisper	O - Goldie	B - Bunny Hop	O - Sprinkles
C - Candy	P - Pop	C - Doodles	P - Peep
D - Clumsy	Q - Pink	D - Marshmellow	Q - Candy Pop
E - Eggy	R - Loco	E - Bubbles	R - Snuggle Bunny
F - Flower	S - Smartie	F - Happy Feet	S - Sunshine
G - Thumper	T - Trixy	G - Baby	T - Sugar Drop
H - Blueberry	U - Carrot	H - Rain Drop	U - Cupcake
I - Purple	V - Daisy	I - Fluffy Tail	V - Sugar Kiss
J - Daffodil	W - Hoppy	J - Sugar Cakes	W - Sparkle Pop
K - Dizzy	X - Shimmer	K - Carrot Cake	X - Carrot Stick
L - Wild	Y - Sweet	L - Junior	Y - Choco Latte
M - Sparkle	Z - Lucky	M - Cotton Tail	Z - Kid

Input

Input will begin with 26 lines. Each line contains: a capital letter, followed by a space, followed by a bunny first name that will replace real first names that start with that letter. The next 26 lines will be in the same format, but specify bunny last names that will replace real last names. You may assume both sets of 26 lines are sorted by capital letter in ascending order. These 52 lines are followed by a line of input that contains a single integer *n*, ($1 \le n \le 1000$), which is the number of real names that follow.

Each real name consists of a single line of input containing two strings separated by a single space representing the real first & last name. Each real first & last name is a string of length *k*, ($1 \le k \le 70$) and contains only letters. The first letter of each real first & last name to be input will always be capitalized.

Output

For each real name in the input, output a line containing: the real name, followed by " = " (space,equals,space), followed by the bunny name resulting from the replacements specified in the first 52 lines of input. There should be only one space between all of the individual string tokens exactly as shown below. You will receive a presentational error if you have multiple spaces between individual string tokens or the equal sign.

Sample Input	Sample Output

```
A Lily
B Whisper
C Candy
D Clumsy
…
Y Sweet
Z Lucky
A Lemon Drop
B Bunny Hop
C Doodles
D Marshmallow
…
Y Choco Latte
Z Kid
2
Bob Allen
LeBron James
```
	```
	Bob Allen = Whisper Lemon Drop
	LeBron James = Wild Sugar Cakes
	```

Triathlon Prediction (prob5)

The Problem

In an effort to promote a healthy lifestyle, Mercer is hosting its first triathlon. A triathlon is a race where competitors swim some distance, then bike another distance, capped off with a run. Different triathlons will set different lengths for each of the segments of the race. For this problem, given the lengths of the three segments of the race in meters and the average speeds of the competitors for each mode of transport in meters/second, create a list sorting the competitors in their expected finishing orders with their times.

Input

The first line of the input will contain a single positive integer, N ($N \leq 100$), representing the number of races for which to make predictions. The data for each race follows. The first line for each race will contain four, space separated positive integers: c ($c \leq 50$), the number of competitors in that race, s ($s \leq 10000$), the number of meters for the swim, b ($b \leq 200000$), the number of meters for the biking portion of the race, and r ($r \leq 50000$), the number of meters for the run at the end of the race. The following c lines will contain information about each of the competitors, one per line. These lines will each have the following space separated information: name of the competitor (an uppercase alphabetic string of 20 or fewer letters), the swimming speed of that competitor in meters/second, the biking speed of that competitor in meters/second and the running speed of that competitor in meters/second. Each of the speeds will be positive real numbers less than or equal to 20 represented to at most two decimal places. It is guaranteed that each of the competitors will have unique finishing times (within a single race) when the number of seconds is truncated. All the finishing times will have a fractional number of seconds greater than .01.

Output

For each race, output a single line header with the following format:

```
Triathlon #k
```

where k is the number of the triathlon, starting with 1.

Each of the following lines should have information about one competitor in the race, sorted by finish time. Each of these should have the name of the competitor, followed by a space, followed by their expected time to finish the race in the following format:

```
H hour(s) M minute(s) S second(s)
```

where H ($0 \le H < 24$), M ($0 \le M < 60$) and S ($0 \le S < 60$) are integers representing their race time. If the number of seconds isn't a whole number, simply truncate it. For example, 45.3 seconds should be written as 45, and 53.98 seconds should be written as 53 seconds.

Sample Input

```
2
3 1000 100000 20000
ALICE 1.3 11.7 3.5
BOB 1.2 10.8 4.0
CHERYL 1.5 12.2 2.9
2 500 50000 8000
DAVINDRA 1.4 13.2 2.9
ISABELLA 1.3 12.8 4.1
```

Sample Output

```
Triathlon #1
ALICE 4 hour(s) 10 minute(s) 30 second(s)
BOB 4 hour(s) 11 minute(s) 32 second(s)
CHERYL 4 hour(s) 22 minute(s) 39 second(s)
Triathlon #2
ISABELLA 1 hour(s) 44 minute(s) 2 second(s)
DAVINDRA 1 hour(s) 55 minute(s) 3 second(s)
```

The Juggler’s Fallacy (prob6)

The Problem

Stephen has run into a slight problem with his friend Bryan. Stephen is a master juggler, and Bryan resents him for it. So much so that he’s taken to the Internet in order to sully Stephen’s good name. Stephen has come across a blog post by someone who he thinks might be Bryan, claiming to be one of the most brilliant jugglers in the world. However, Stephen suspects that some of the sequences this “@B-ri” claims to be able to juggle just aren’t possible. He’s enlisted your help to solve his problem.

A juggling sequence consists of a finite sequence of positive integers. Each integer in the sequence consists of a throw of one of the balls, and its value represents how many “beats” until the juggler catches the ball. Essentially any sequence of integers is juggleable as long as the juggler only ever has to catch and throw one ball on a given beat. The only special case is the case of a 0 beat throw, which is a pause and consists of neither a catch or a throw.

Consider the juggling sequence *3 3 3*. Even though this is a finite sequence, this continues on for forever, and encodes the infinite sequence of throws of duration 3. With this sequence, the juggler throws their first ball at time step 0, and catches that ball at time step 3. At time step 3, the juggler catches the ball thrown at time step 0 and throws another ball with a duration of 3. This sequence is able to be juggled since each throw of 3 will land on a unique time step.

However, the sequence *2 1* is not able to be juggled. At time step 0, the juggler throws a ball that will land at time step 2. At time step 1, the juggler will throw a ball that will also land at time step 2. This means the juggler needs to catch both balls, *and* throw a ball with a duration of 2. This is not physically possible, and thus the sequence is not valid.

Input

Input will consist of a single integer ***N*** specifying the number of juggling sequences to follow. Each of the following ***N*** lines will consist of no more than 25 integers, each separated by a single space. Since most jugglers can only manage a relatively small number of balls, you can assume that each integer is in the range [0, 25].

Output

Output will consist of ***N*** lines. Each line should output the sequence, followed by a ‘ - ‘, then followed by a statement claiming whether the sequence can or cannot be juggled.

Sample Input

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

Sample Output

```
3 3 3 - can be juggled
5 0 1 - can be juggled
2 1 - cannot be juggled
4 4 1 3 - can be juggled
```

Downhill Marble Labyrinth (prob7)

The Problem

Marble labyrinth is a handheld dexterity game that consists of a maze with holes sitting on top of a box and a small marble. The objective is to guide the marble through the maze without letting it fall into any of the holes. Consider a variation of the game where we have a landscape instead of the maze and the marble is only allowed to move to an adjacent position either horizontally or vertically that is lower in elevation.

The landscape is represented in the form of a 2 dimensional grid with m rows and n columns. The number in each grid is a non-negative integer that gives the elevation of the grid. Please count the total number of valid paths for the marble to go from the top left position to the bottom right. For example, there are two paths for the landscape given by:

7	6	5
9	4	3
1	8	2

Input

The first line of input contains the total number of test cases. The next several lines describe each of these test cases. Each test case contains m and n on the first line, where 1 ≤ m, n ≤ 1000. The following m lines contain exactly n non-negative unique integers, each in the range [1, 1000000000] that represent elevations.

Output

For each of the test cases, please return the total number of valid paths from the top left to the bottom right positions. All results should be modulo 1000.

Sample Input

```
2
3 3
7 6 5
9 4 3
1 8 2
2 2
4 3
2 1
```

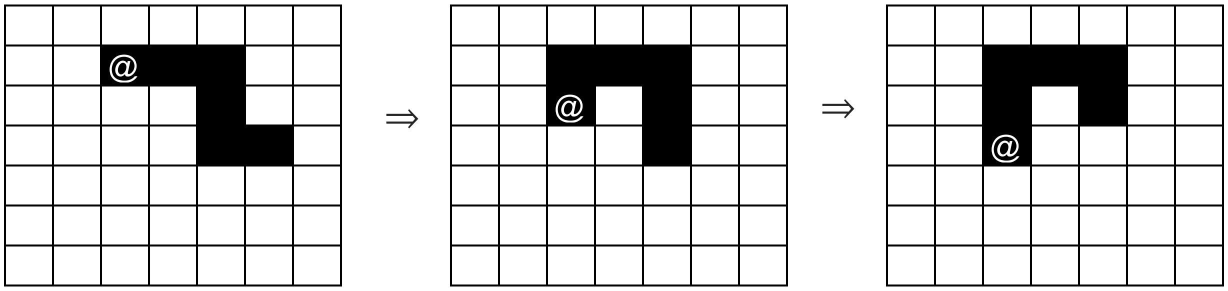
Sample Output

```
2
2
```

Snake Game (prob8)

The Problem

Snake is a video game in which a player controls a sequence of squares (the snake) on a limited 2 dimensional grid (field). Each square in a sequence is 4-connected to the previous one (they share a side on the grid). On each time step (e.g. each second) the first square of the sequence (the head of the snake) moves to one of 4 possible directions (left/right/up/down) and the rest of the snake follows it, i.e. the second square moves to the previous position of the head, the next square takes the old position of the second square and so on. The previous position of the last square of the snake becomes free.



Snake cannot go beyond the boundaries of the field, cannot go to cells with obstacles or over its own body, i.e. the head of the snake cannot move to a square that will be occupied by another part of its body after the move is complete. Note, that if the snake is longer than 3 squares it is possible for the head to go to a cell where the tail currently is, because the tail will also move and make its current position available.
In this problem the goal is to get to the specified "magical" cell. Of course playing games for you as a programmer is boring. It is much more fun to code. Thus let's write a program to find the minimal number of steps the snake needs to reach the magical square.

Input

The input starts with a single integer T ($1 \leq T \leq 30$) - the number of test cases.

The first line of each test case contains 3 integers N, M, K. Where N and M are dimensions (number of rows and columns) of the field ($1 \leq N, M \leq 10$) and K is the length of the snake ($1 \leq K \leq 10$).

The next N lines describes the field and have M characters each. Each character can be:

`'.'` - a cell is free

`'x'` - a cell is a magical one. There will be exactly one magical square on the board.

`'#'` - a cell contains an obstacle and the snake cannot move to it

`'0', .., '9'` - parts of the snake. `'0'` is the head of the snake and the other parts are numbered sequentially. It is guaranteed, that the input is correct and the snake is "connected", i.e. head of the snake `'0'` is 4-connected to `'1'`, which is 4-connected to `'2'`, etc.

Output

For each i-th test case output "`Game #i: <res>`" (without quotes), where i is the 1-based number of test case and <res> is the minimal number of steps to reach the magical square or -1 if it is impossible (see sample output).

Sample Input	**Sample Output**

`5`
`3 2 2`
`0.`
`1.`
`.x`
`3 3 2`
`.x.`
`##.`
`10.`
`5 5 5`
`....x`
`..432`
`....1`
`....0`
`.....`
`2 3 4`
`x10`
`.23`
`1 3 2`
`x10`
	`Game #1: 3`
	`Game #2: 4`
	`Game #3: 5`
	`Game #4: 4`
	`Game #5: -1`

Tree Planting (prob9)

The Problem

You are the owner of an apple orchard, and you are trying to figure out the best locations to plant your apple trees. On this orchard, there are *N* rows where you can plant *T* apple trees. Orchard row *i* has width w_i and a tree can be planted on row *i* in any location in the range $[0, w_i]$.

For planting the trees, you want them to have enough space so that each tree has proper access to soil nutrients and sunlight. If they are planted too close to another tree (or to the end of a row), then they may not receive enough nutrients and sunlight to survive. Define the spacing of a tree to be the minimum of

1) the distance of this tree to another tree in the same row
2) the distance of this tree to either end of its row

The spacing of the entire planting is then the minimum spacing over all trees. We assume that there is adequate spacing between the orchard rows so that trees in separate rows do not compete for nutrients or sunlight. The objective of this problem is to determine the maximum spacing for a planting, given a set of rows and number of trees. The bigger the spacing, the better chance that all the trees survive!

Input

The first line of input is a number *t*, specifying the number of test cases. For each test case, the first line will be two integers, *N* and *T*, where $1 \le N \le 10^6$ and $1 \le T \le 10^9$. On the second line, *N* numbers follow, specifying the width of each orchard row to 1 decimal place. The width of each row is at most 10^6.

Output

For each test case, print the maximum spacing achievable for a planting, rounded to 3 decimal places. Each test case should be followed by a newline character.

Sample Input	Sample Output

`2`
`1 3`
`4.0`
`2 4`
`3.0 5.5`
	`1.000`
	`1.375`

Sums of Squares (prob10)

The Problem

Vivian likes numbers that can be represented as sums of distinct squares, but she doesn't like consecutive squares to be in her sums. For example, she likes 117 because

$$1^2 + 4^2 + 6^2 + 8^2 = 117$$

Then she realizes that she can obtain the same sum with a different set of squares, none of which are consecutive:

$$1^2 + 4^2 + 10^2 = 117 \quad \text{or} \quad 6^2 + 9^2 = 117$$

Now, Vivian wants to know, given some positive integer, K, how many sets of perfect squares add up to K, where no two of the squares are squares of consecutive integers. Write a program to answer her query!

Input

The first line of the input will contain a single positive integer, N (N ≤ 10000), representing the number of test cases. Each of the following N lines will contain a single positive integer, K (K ≤ 10000), representing the queried integer.

Output

For each input K, output on a line by itself the number of ways K can be represented as the sum of a set of squares, none of which are consecutive.

Sample Input

```
4
117
200
53
5
```

Sample Output

```
3
3
2
0
```

Elliptical Reflections (prob11)

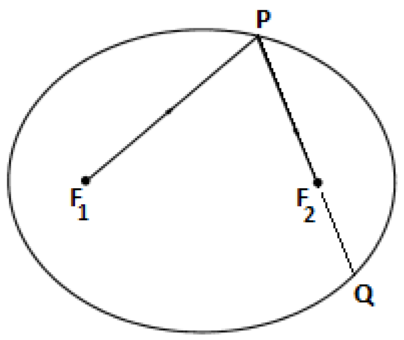
The Problem

An ellipse has the interesting property that if one emits a signal from one of its foci, it will bounce off of a point on the ellipse and the signal is reflected through the other focus. See the figure below. For this problem, you will be given a focus F_1, a point P on the ellipse, and the other focus F_2. You are to determine the next point Q, on the ellipse, that will be hit when a signal is emitted from the first focus F_1, it bounces off the ellipse at P and then passes through F_2.

Recall, if P is a point on an ellipse having foci F_1 and F_2, then $|PF_1| + |PF_2| = 2a$, $|F_1F_2| = 2c$ and the equation of the ellipse having center (h, k) is

$$(a^2 - c^2)(x - h)^2 + a^2 (y - k)^2 = a^2 (a^2 - c^2)$$

if the major axis, the line through F_1 and F_2, is horizontal.



Input

The input for each case will be on a line by itself. It will consist of six numbers x_1, y_1, x_0, y_0, x_2, y_2 where $P = (x_0, y_0)$, $F_1 = (x_1, y_1)$, and $F_2 = (x_2, y_2)$. The input will be terminated by a line of input where $F_1 = F_2 =$ the origin. The last line is not to be processed. You may assume for all the other lines of input that $F_1 \neq F_2$, that the major axis of the ellipsis is either vertical or horizontal, and that P is not on the segment F_1F_2. Also, the absolute value of any coordinate will not exceed 100.

Output

For each line of output, output the coordinates of the point Q rounded to four decimal places.

Sample Input

```
-5 0 19 5 5 0
-5 0 -17.800166 -8.142916 5 0
5 0 -19 5 -5 0
5 0 17.800166 -8.142916 -5 0
-3 4 21 9 7 4
0 -5 5 19 0 5
4 -3 9 21 4 7
0 0 0 0 0 0
```

Sample Output

```
-17.8002 -8.1429
19.0000 5.0000
17.8002 -8.1429
-19.0000 5.0000
-15.8002 -4.1429
-8.1429 -17.8002
-4.1429 -15.8002
```

Fun With Fortran IV (prob12)

The Problem

As everyone knows (or should know), Fortran is the ancestor of all high-level programming languages. The most significant version was Fortran IV, which is the version we shall consider here. Fortran had a number of interesting "quirks" when compared to modern languages. For example, spaces were completely insignificant. Thus, the statements "`GOTO 23`", "`GO TO 23`" and "`G O T O 2 3`" were all equivalent (and valid). Also, there were no reserved keywords, so "`IF`," "`DO`," etc. could be used as variable names (or as a prefix to a variable name), and variables did not have to be declared. These three aspects of the language combined to cause a very famous bug: a `DO` statement that should have been

```
      DO 97 I = 1,5
```

was actually coded as

```
      DO 97 I = 1+5
```

which was recognized as a valid assignment statement (assigning the value 6 to the variable `DO97I`) instead of looping 5 times, and an experimental rocket blew up.

Your job is to write a program that parses Fortran IV statements. The Fortran subset that you are to work with is described by the following grammar:

⟨*statement*⟩	::= ⟨*dimensionstatement*⟩ | ⟨*assignmentstatement*⟩
	| ⟨*ifstatement*⟩ | ⟨*dostatement*⟩ | ⟨*gotostatement*⟩
	| ⟨*endstatement*⟩
⟨*dimensionstatement*⟩	::= ‘`DIMENSION`’ ⟨*dimensionlist*⟩
⟨*dimensionlist*⟩	::= ⟨*dimensionitem*⟩
	| ⟨*dimensionitem*⟩ ‘`,`’ ⟨*dimensionlist*⟩
⟨*dimensionitem*⟩	::= ⟨*identifier*⟩ ‘`(`’ ⟨*integerconstant*⟩ ‘`)`’
⟨*assignmentstatement*⟩	::= ⟨*variable*⟩ ‘=’ ⟨*expression*⟩
⟨*variable*⟩	::= ⟨*identifier*⟩ | ⟨*identifier*⟩ ‘`(`’ ⟨*expression*⟩ ‘`)`’
⟨*expression*⟩	::= ⟨*integerconstant*⟩ | ⟨*variable*⟩ | ‘`(`’ ⟨*expression*⟩ ‘`)`’
	| ⟨*expression*⟩ ⟨*operator*⟩ ⟨*expression*⟩
⟨*operator*⟩	::= ‘`+`’ | ‘`-`’ | ‘`*`’ | ‘`/`’ | ‘`**`’
⟨*ifstatement*⟩	::= ‘`IF`’ ‘`(`’ ⟨*expression*⟩ ‘`)`’ ⟨*integerconstant*⟩ ‘`,`’ ⟨*integerconstant*⟩ ‘`,`’ ⟨*integerconstant*⟩
⟨*dostatement*⟩	::= ‘`DO`’ ⟨*integerconstant*⟩ ⟨*identifier*⟩ ‘=’ ⟨*dolist*⟩
⟨*dolist*⟩	::= ⟨*doitem*⟩ ‘`,`’ ⟨*doitem*⟩
	| ⟨*doitem*⟩ ‘`,`’ ⟨*doitem*⟩ ‘`,`’ ⟨*doitem*⟩
⟨*doitem*⟩	::= ⟨*integerconstant*⟩ | ⟨*identifier*⟩
⟨*gotostatement*⟩	::= ‘`GOTO`’ ⟨*integerconstant*⟩
⟨*endstatement*⟩	::= ‘`END`’
⟨*identifier*⟩	::= ⟨*letter*⟩ | ⟨*identifier*⟩ ⟨*letter*⟩ | ⟨*identifier*⟩ ⟨*digit*⟩
⟨*integerconstant*⟩	::= ⟨*digit*⟩ | ⟨*integerconstant*⟩ ⟨*digit*⟩
⟨*letter*⟩	::= ‘`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`’
⟨*digit*⟩	::= ‘`0`’ | ‘`1`’ | ‘`2`’ | ‘`3`’ | ‘`4`’ | ‘`5`’ | ‘`6`’ | ‘`7`’ | ‘`8`’ | ‘`9`’

Input

Each input line is a non-empty string representing a potential Fortran statement. No line will be longer than 72 characters, and will consist only of characters in the standard Fortran characters set: uppercase letters, digits, spaces, and the “special” characters

```
=+-*/(),.$
```

Your program should exit after processing the `END` statement.

Output

For each line of input (including the `END` statement), there will be one line of output, stating which kind of Fortran statement was recognized: a ‘`#`’ sign followed by the input line count; a space; one of the words “`ASSIGNMENT`”, “`DIMENSION`”, “`DO`”, “`END`”, “`GOTO`”, “`IF`” or “`INVALID`”; a space; and the word “`STATEMENT`”.

Sample Input

```
A = 9
X(3) = (5 + 7) ** 9
DIMENSION A(3), B(5), C(100)
DO 97 I = 1,5
DO 97 I = 1+5
GO TO 23
IF(X - 3)21,32,33
IF(X - 3) = 5 000 000
GOTO SOMEWHERE
DIMENSION X(5 + 3)
END
```

Sample Output

```
#1: ASSIGNMENT STATEMENT
#2: ASSIGNMENT STATEMENT
#3: DIMENSION STATEMENT
#4: DO STATEMENT
#5: ASSIGNMENT STATEMENT
#6: GOTO STATEMENT
#7: IF STATEMENT
#8: ASSIGNMENT STATEMENT
#9: INVALID STATEMENT
#10: INVALID STATEMENT
#11: END STATEMENT
```

Planet Surveying (prob15)

The Problem

You've just been hired by Hubert Farnsworth to perform planetary surveys in order to optimize shipping on a planet. However, for your first task, he's given you some stats on planets and wants you to give some surveying estimates before you fly out to uncharted regions of the galaxy and perform the real surveying.

The method in which the professor wants you to survey is as follows: Start at the South pole of the planet, spiraling upwards to the North pole. When spiraling, he wants you to keep your survey ship to maintain a constant slope relative to latitude and longitude. He gives this as a ratio.

For instance, if he says use a slope of k = 2, then for every degree right of longitude the ship goes, the ship must also go up 2 degrees latitude. Also, you can assume the planets are perfect spheres.

Input

Professor Farnsworth has given you a data file describing a couple of planets he'd like for you to estimate.

The first row has a single integer N (up to 2000).

Every N lines after that have two decimal numbers.

The first is R (1 <= R <= 100), the radius of the planet in 1000's of kilometers.

The second is k (1 <= k <= 20), the slope to use.

Output

For your output, you should print a single number, an estimate of the length of the path in 1000's of kilometers, formatted to three decimal places.

Sample Input

```
2
10.0 1.0
20.0 2.0
```

Sample Output

```
38.202
66.592
```