1- MPG
Input File: MpgIn.txt

A car's computer system displays the total miles traveled (DMT) and the average miles per gallon (MPG), since the last time the display was reset. Every 30 minutes when the car is running, the system records the miles traveled (RMT) and the gallons of fuel used (G) during the past 30 minutes, and then updates the displays.

Given the current status of the display (DMT and MPG), and the recorded distance traveled (RMT) and the gallons of fuel consumed (G) during the last 30 minutes, compute the new values of the displays: the new displayed miles traveled (NDMT) and the new average miles per gallon (NMPG) .

Note: NDMT = DMT + RMT and NMPG = NDMT / (DMT / MPG + G)

Inputs:
There will be one line of input that contains four integers separated by a space. These values will be the displayed miles traveled (DMT), followed by the displayed miles per gallon (MPG), followed by the recorded miles traveled (RMT) and the fuel consumed (G) over the past 30 minutes.

Outputs:
There will be one line of output that contains the new displayed value of the total miles traveled (NDMT), followed by the new displayed value of the average miles per gallon (NMPG). These two computed values, NDMT and NMPG, will always be integers and they will be annotated exactly as shown below.

Sample Input
```
10 20 5 1
```

Sample Output:
```
15 miles 10 mpg
```

2- Intersection
Input File: IntersectionIn.txt

A driverless car is approaching an isolated intersection, and its traffic sensors indicate that there are no other cars in, or approaching, the intersection from any direction. Under these circumstances when the light is green, the car can proceed straight through the intersection, or signal a right or left turn and then proceed. When the light is red the car must stop, or it can make a right turn after stopping and signaling.

This gives three choices on a *green* light represented by the three alternative command codes: P, or RT, or LT that are sent to the car's virtual driver defined as,

P - Proceed straight; RT - signal Right then Turn; LT- signal Left then Turn.

Alternately, there are two other command codes on a *red* light: S, or SRT defined as,

S- Stop ; SRT - Stop, signal Right then Turn.

Given the status of the traffic light (red or green) as the car approaches the intersection, and the direction the car will take through the intersection (straight, left, or right) , your task is to determine the car's command choice.

Inputs:
The first line of input will be the number of intersections to consider. This will followed by line one of input per intersection that contains two integers. The first integer represents the color of the traffic light: 1 for green, 2 for red. The second integer represents the car's intended direction through the intersection: 1 for straight, 2 for left turn, 3 for right turn. All inputs on a line are separated by a space.

Outputs:
There will be one line of output per intersection that contains command code transmitted to the virtual driver at that intersection. All outputs will be in upper case.

Sample Inputs

```
6
1 2
1 1
1 3
2 2
2 1
2 3
```

Sample Outputs

```
LT
P
RT
S
S
SRT
```

3- Windy Walk
Input File: WindyWalkIn.txt

Logan lives in a windy neighborhood, and walks outdoors for exercise. On some days, the wind gusts are so strong that they drive him back one, or more, steps per gust. Assuming his forward and backward steps are exactly the same length and each step takes one second, calculate the time it takes him to reach his destination given the number of forward steps between his starting point and his destination, **d**, and his forward and backward step sequence.

Forward motions in the step sequence are represented by positive integers, and backward motions are represented by negative integers. For example, if Logan's walk consisted of 2 forward steps, 1 backward step, 1 forward step, and finally 2 backward steps the sequence would be: 2, -1, 1, -2. The sequence may contain more steps than are needed to reach the destination, because some days Logan walks beyond the destination.

Inputs:
The first line of input will be the number of walks Logan will take, **n**, followed by one group of inputs for each walk. The first line in a grouping will contain one integer that is the number of forward steps between his starting point and his destination, **d**. The second line of input in the grouping will contain a sequence of integers that represents his forward and backward step sequence for the walk. All inputs on a line are separated by a space.

Outputs:
There will be one line output per walk that contains one integer that represents the exact time, in seconds, that it takes Logan to reach his destination for the first time.

Sample Inputs
```
2
5
3 -2 -1 4 -1 1 -1 1 1
8
-3 3 3 -2 4 -1 1 2 3 -3 1 1 1 2
```

Sample Outputs:
```
15
20
```

4- Numbers Game
Input File: NumbersGameIn.txt

An encrypted message is sent to you in which the letters of each word have been randomly rearranged, and the spaces between words have been eliminated. For example: `osuurcraetemnpf`. Then a decryption key is sent to you that contains a sequence of integers, each separated by a space. The 1st integer in this sequence is the character number in the encrypted message that will be the 1st character in the decrypted message, the 2nd integer is the character number in the encrypted message that will be the 2nd character in the decrypted message, etc. Integers greater than 20 insert a space into the decrypted message.

For example, using the integer sequence 6 1 12 14 10 4 7 9 2 **99** 8 5 11 **21** 15 3 13 to decrypt the message `osuurcraetemnpf` yields the decrypted message `computers are fun`.

Your task is to produce the decrypted message given the encrypted message and the decryption key.

Inputs:
The first line of input will be the number of messages to decrypt, followed by three lines of input for each message. The first of these lines will contain the encrypted message. The second line will contain an integer that represents the number of integers, **n**, in the decryption key. The third line will contain a sequence of **n** integers, each separated by a space, which is the decryption key.

Outputs:
There will be one line of output per message that is the decrypted message.

Sample Inputs
```
2
osuurcraetemnpf
17
6 1 12 14 4 10 9 7 2 99 8 5 11 21 15 3 13
uebso!emnaemsrdlvg
20
9 1 12 3 11 14 4 22 18 10 8 7 21 13 5 16 17 2 15 6
```

Sample Outputs
```
computers are fun
numbers game solved!
```

5- Math Syntax
Input File: MathSyntaxIn.txt

Nothing annoys Professor Plumb more than students who write improperly formed infix math expressions on his examinations. To avoid this annoyance, he has asked you to grade the exams, and you have decided to automate the grading.

According to the professor, infix math expressions always begin and end with a possibly signed numeric value (e.g., `-1.1 7.5`) . In between these two numerics, there must be one or more pairs of a math operator followed by a numeric value (e.g., `+ -2.34`) , and the numeric value can be a signed value. The four allowable operators are `+`, `–`, `*` and `/`. All entities (numerics and operators) must be separated by *exactly one* space. For example:

```
  -1.1 * -2.34 - 7.5     and      3 * 3.14159 – 18 + -6.6
```

are both valid infix expressions. The following four expressions are all invalid, and a carat (**^**) has been shown below the first invalid character in the expression when examining the expression from left to right.

```
1.1 +   7.5  and  1 + - 2  and  3.14 * 6.8 811   and   / 2 + 3
       ^              ^                     ^          ^
     extra space   extra                  extra numeric non-numeric
```

Inputs:
The first line of input will be the number of math expression on the examination, **n**, followed by the math expressions to grade, each one on a separate line.

Outputs:
There will be one line output per math expression that contains the word `correct` for valid math expressions. For an invalid expression, output the character position (e.g., 1 for the first character, 2 for the second character, …) of the first invalid character in the math expression when examining the expression from left to right. However, when the invalid expression ends with an operator, output the integer that is one more than the number of characters in the math expression (see the last sample input and output below.).

Sample Inputs

```
8
-1.1 * -2.34 - 7.5
3 * 3.14159 - 18 + -6.6
1.1 +   7.5
1 + - 2
3.14 * 6.8 811
3.14 x 6 / 6.845 8
/ 2 + 3
2 + 3 +
```

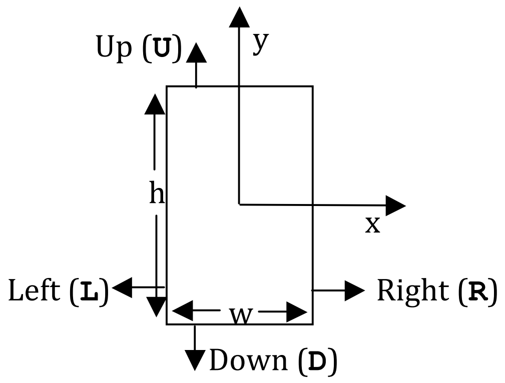
Sample Outputs:

```
correct
correct
7
5
12
6
1
8
```

6- Dart Evader
Input File: DartEavderIn.txt

Breanne and her friend, Luke Flystalker, like to play the game Dart Evader. In this game Breanne stands behind a rectangular target that she is holding, which faces Luke, and remains perfectly still until Flystalker throws a dart at the target. Then she can move the target vertically up or down, or horizontally to Skywalker's left or right, in attempt to prevent the dart from striking the target. If the dart hits the target or its edge, Flystalker receives a point. Otherwise, Breanne receive a point.

As shown below, a two-dimensional Cartesian coordinate system is associated with the target, whose origin is at the center of the target, and whose positive x and y directions are to Luke's right and upward respectively. The target's width is **`w`** inches and its height is **`h`** inches.



Your task is to determine the score of the game, given the number of darts thrown by Flystalker, the (x, y) coordinates of each dart's aiming point, the width and height of the target, and the maximum distance Breanne can move the target up (**`U`**), down (**`D`**), left (**`L`**) and right (**`R`**). The units of all coordinates and distances are inches.

Inputs:
The first line of input will be the number of games played, **`n`**, followed by one group of inputs for each game. The first line in a grouping will contain six integers: the target's width (**`w`**), followed by its height (**`h`**), followed by the maximum distance the target can be moved in the upward (**`U`**), downward (**`D`**), left (**`L`**) and right (**`R`**) directions. The second line of input in the grouping will contain one integer, which is the number of darts (**`d`**) thrown in this game. This will be follow by **`d`** lines (one per dart) that contain two *real numbers* per line that represent the x and y coordinates of a dart's aiming point. All inputs on a line will be separated by a space, and the units of the aiming point coordinates, the target's dimensions, and the maximum distances the target can be moved is inches.

Outputs:

There will be one line of output per game that contains two integers separated by a space. The first integer will be Breanne's score, and the second integer will be Luke's Flystalker's score.

(Sample inputs and outputs on next page)

Sample Inputs

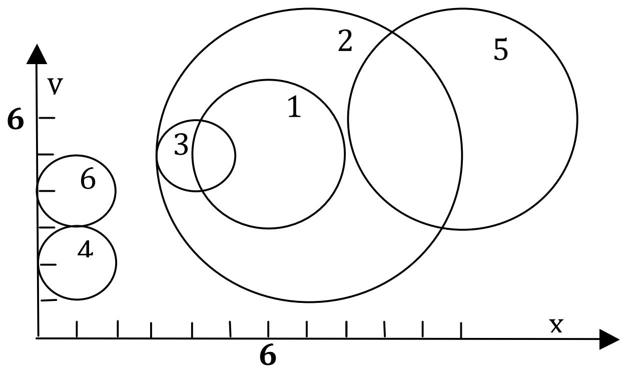
```
1
5 10 3 4 1 2
10
0.0 0.0
0.0 -2.01
0.0 -2.00
-3.0 0.0
5.0 0.0
0.0 5.0
0.0 -4.0
-0.51 0.0
1.50 0.0
0.0 1.01
```

Sample Outputs:

```
7 3
```

7- Inside Outside
Input File: InsideOutsideIn.txt

Nora has randomly distributed a collection of **n** circular disks, of various radii, onto a gymnasium floor. Each disk has a unique number printed on it, in the range from 1 to **n.** Given a disk number of interest, **d**, your task is to determine all of the disks that are completely inside of its perimeter, and all of the disks that are completely outside of it (share none of their area with disk **d**). For the disks shown below with **d** = 2, disks 1 and 3 are completely inside of disk 2, and disks 4, 5, and 6 are completely outside of disk 2. Note: a disk that is tangent to another disk can either be completely inside the other disk (e.g., disks 3 and 2 below), or completely outside of the other disk (e.g., disks 4 and 6 below).



Inputs:
The first line of input will be the number of gymnasium floors to consider, followed by one group of inputs for each floor. The first line in a grouping will contain two integers that represent the number of disks on the floor, **n**, followed by the disk number of interest, **d**. This will be followed by one line of input per disk that describes disk 1, disk 2, …, and disk **n**. Each of these **n** lines contains three integers in this order: the x followed by the y coordinate of the center of the disk, followed by its radius. All inputs on a line are separated by a space.

Outputs:
There will be one line of output per gymnasium floor that contains the disk numbers *inside* the disk of interest in ascending order, followed by the disk numbers *outside* the disk of interest in ascending order. The outputs on a line will be separated by a space.

Sample Inputs (see the figure above)

```
2
6 2
6 5 2
7 5 4
4 5 1
1 2 1
11 6 3
1 4 1
6 6
6 5 2
7 5 4
```

```
4 5 1
1 2 1
11 6 3
1 4 1
```

Sample Outputs

```
1 3 4 6
1 2 3 4 5
```