Display Tester

Program Name: display.java　　　Input File: <none>

On a computer system with multiple displays, it can often be useful to have a program to help identify which is which. These programs display a known message that the technician will recognize.

Write a program that will print a text message.

Input

This program requires no input.

Output

Output the statement, “This is the current display.”

Example Output To Screen

```
This is the current display.
```

Are we there yet?

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

Some vehicle dashboards display useful information such as estimated time left in a trip. You are tasked with writing software that, given odometer and clock readings for three legs of a 200 mile trip, will display the estimated time left in the trip.

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 three pairs of integers. Each pair represents the odometer and clock reading at the end of the corresponding leg of the trip. Assume the following:

- the odometer and clock both start at zero when the trip begins
- each leg will cover at least 1 mile and take at least 1 second
- the total trip will not take more than 5 hours

Output

For each data set in the input, display three integers on their own line in the format "*X Y Z*"where *X*, *Y*, and *Z* are the estimated seconds (rounded to the nearest integer) left in the trip at the end of each leg. Base this estimate on the number of miles left in the trip and the average speed traveled so far in the trip.

Example Input File

```
3
100 3600 150 7200 175 14400
1 1 2 2 3 3
1 1 2 3 3 6
```

Example Output To Screen

```
3600 2400 2057
199 198 197
199 297 394
```

Big Numbers

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

Your programming teacher has just asked you to write a program to manipulate, “big numbers.” Since you're feeling snarky, you decide to interpret the instructions literally by writing a program that displays numbers in a really big font:

```
  1   22222 33333 4   4 55555 66666 77777 88888 99999 00000
 11       2     3 4   4 5     6         7 8   8 9   9 0   0
  1   22222   333 44444 55555 66666     7 88888 99999 0   0
  1   2         3     4     5 6   6     7 8   8     9 0   0
11111 22222 33333     4 55555 66666     7 88888 99999 00000
```

Input

The first line of input will consist of a single integer, *n*, indicating how many numbers are to be displayed. The following *n* lines will each contain a single non-negative integer that is to be displayed in the really big font. These integers will consist of up to ten digits.

Output

Output each of the numbers on its own set of 5 lines, using the format displayed in the introduction. Note that each digit is 5x5 and that they are separated by a column of spaces of width 1.

Example Input File

```
3
1234567890
0
1
```

Example Output To Screen

```
  1   22222 33333 4   4 55555 66666 77777 88888 99999 00000
 11       2     3 4   4 5     6         7 8   8 9   9 0   0
  1   22222   333 44444 55555 66666     7 88888 99999 0   0
  1   2         3     4     5 6   6     7 8   8     9 0   0
11111 22222 33333     4 55555 66666     7 88888 99999 00000
00000
0   0
0   0
0   0
00000
  1
 11
  1
  1
11111
```

Gravity Rocks!

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

NASA has recently discovered how to create artificial gravity. As an intern at the space agency, it is your job to write a program that simulates the effects of artificial gravity on previously weightless rocks. Since full three dimensional models are very complicated, you start by writing a quick prototype that only considers two dimensions.

Write a program that will visually display the effect of gravity on a set of floating rocks in a 2-dimensional simulation.

Input

The first line of input contains a single integer, *n*, indicating the number of data sets to process. Each data set consists of:

1. Ten lines of 10 characters each representing a 10x10 grid of space containing some number of floating rocks. Each rock is represented by a single pound character ('#'), while empty space is represented by a period ('.').
2. A line containing one of the following characters indicating the direction of the artificial gravity field:
   U - up
   D - down
   L - left
   R – right

Output

For each data set in the input, first print the message, “Data Set #X” where X is 1 for the first data set, 2 for the second, etc. Then output the 10x10 grid of space, showing the new resting position of rocks after the indicated artificial gravity field has been applied. For instance, if an 'R'-type gravity field is applied to the rocks, they will all move as far as possible to the right of the 10x10 field, each stacking on top of any other blocks that might be in the way.

Don't forget: Each rock is a single character and moves independently.

Example Input File

```
2
.#........
..........
.#.#...#..
..........
.........#
....###...
##....#...
...#.#...#
.......#..
..........
L
..........
.......#..
...#.#...#
##....#...
....###...
.........#
..........
.#.#...#..
..........
.#........
U
```

Example Output To Screen

```
Data Set #1
#.........
..........
###.......
..........
#.........
###.......
###.......
###.......
#.........
..........
Data Set #2
##.#####.#
.#.#.###.#
.#........
..........
..........
..........
..........
..........
..........
..........
```

Who Knows Whom?

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

In police work, it is often useful to have a list of known associates for a criminal. Having this information for multiple criminals helps officers understand social networks in the areas where they work. For instance, if person A is a known associate of person B, and person B is a known associate of person C, there is an indirect connection between persons A and C.

Write a program that can determine whether or not given pairs of people have a connection (whether direct or indirect) based on a list of known associations.

Input

The first line of the input file will contain two integers, *m* and *n*, in the range 1 to 20.
The next *m* lines will each contain one pair of names with length in the range of 1 to 20 characters. These are people who are known to be associates.
The next *n* lines will each contain one pair of names with length in the range of 1 to 20 characters. These are people for which the program must determine if a known association exists.

Output

For each of the name pairs in the second list, output a single line. If the pair has a connection print, “name1 is connected to name2.” Otherwise print, “name1 has no known connection to name2.”

Example Input File

```
6 4
Carol Mike
James FashionVictim
Hershey Buddy
Tim James
Candice Mike
Tim Hershey
Mike Carol
Carol Candice
Mike FashionVictim
Buddy James
```

Example Output To Screen

```
Mike is connected to Carol
Carol is connected to Candice
Mike has no known connection to FashionVictim
Buddy is connected to James
```

Robofence

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

You are a rich, lazy farmer. But just because you're a retired dot-com billionaire doesn't mean that you want to spend too much money hiring professionals to put up a fence on your property. So you've done what any reasonable farmer would do, built a 2-ton fencing robot from a slew of Lego Mindstorms sets.

The robot's programming is mostly complete, but you want to add one final feature.

Write a program that will review the robot's path and report the total number of yards of fencing used and the total area enclosed by the fence.

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 line containing a single integer, 4 <= *m* <= 20, indicating how many straight stretches of fence the robot will be putting up.
2. A line containing *m* integers separated by *m*-1 letters. Each integer is the number of yards long that a stretch of fence will be. Each letter will indicate the direction the robot will turn between two stretches of fence (either 'L' – left, or 'R' – right). Turns are always exactly 90 degrees.

Fence laying instructions will always form an enclosed fence (i.e., the robot will always end up where it started), and the fence will never cross itself. Also, the entire fenced area will be coverable by a 50x50 square.

Output

For each dataset in the input, output the statement, "Fence X is Y yards long and encloses an area of Z square yard(s)." on its own line. X will be 1 for the first dataset, 2 for the second, etc. Y and Z are the values you will calculate.

Example Input File

```
2
4
2 L 2 L 2 L 2
8
3 L 6 L 3 L 2 R 2 L 2 L 2 R 2
```

Example Output To Screen

```
Fence 1 is 8 yards long and encloses an area of 4 square yard(s).
Fence 2 is 22 yards long and encloses an area of 22 square yard(s).
```