Manitoba High School Programming Contest

University of Manitoba

May 22, 2009 1:30-4:30 PM

Problem Packs

Notes:

- All input should be read from standard input (the keyboard) and written to standard output (the screen).
- Any programming language resources are allowed.

Problem 1 – Humidex

Humidex is a measurement made to give people an idea of how hot it feels out, as opposed to how hot it actually is: on days with high temperature and high humidity, people often perceive that it feels hotter than it actually is. The formula for calculating the humidex is:

$$R = 6.11 * e^{[5417.7530 * ((1/273.16) - (1/(273.16 + D)))]}$$
$$H = (0.5555) * (R - 10.0)$$
$$humidex = T + H$$

Here e is Euler's number (e = 2.71828182845904523536....), D is the dew point and T is the air temperature (both in °C). Your job is to write a simple program which accepts the two inputs above and prints the humidex value.

Input

The input for a run of this program is a single line which consists of two floating point numbers. The first number represents D in the formula, while the second represents T.

Output

For the input, write out "`The humidex is`" followed by the humidex calculation, rounded to one decimal point, followed by a period. If the number after the decimal point after rounding is zero, it should be printed.

Sample Input	Sample Output
`16 30`	`The humidex is 34.6.`

Judging Data for Problem 1

Note: since this problem does not involve iteration, your program will be run separately on each of the input files listed below.

```
File 1:
16 30

File 2:
10 30

File 3:
20 30

File 4:
20 33

File 5:
0 20

File 6:
-1 15

File 7:
20.3 38
```

Problem 2 – Knuth's Up Arrow Notation

In 1976, Donald Knuth introduced the up arrow notation for denoting large numbers. The up arrow allows us to easily express large numbers by using towers of exponents. (To make things readable, let's denote a^b by `a^b.`) The expression a↑↑b where a and b are numbers means

```
a^(a^(a^(a^.....)))
```

where **there are b-1 exponentiations and b copies of the number a**. Thus, the value of the expression 3↑↑3 is 3^(3^3) = 3^27. As you can imagine, numbers grow really quickly using the up arrow notation.

Write a program which computes the result of an up arrow expression. The results of the up arrow operation will be small enough to fit in a long (64-bit) data type.

Input

The input consists of a first line which contains a single number *n*, the number of test cases. All other lines contain a pair of numbers *a b*, which represents a↑↑b. The values of *a* and *b* are guaranteed to both positive numbers.

Output

For each case, print out either the value of a↑↑b on the line.

Sample Input	Sample Output
3 3 2 2 3 2 4	27 16 65536

Judging Data for Problem 2

```
30
1 1
1 2
1 3
1 4
1 5
1 1000
2 1
2 2
2 3
2 4
3 1
3 2
3 3
4 1
4 2
5 1
5 2
6 1
6 2
7 2
8 2
9 2
10 2
11 2
12 2
13 2
14 2
15 2
1000 1
10000 1
```

Problem 3 – Snakes and Ladders

An array of integers (positive and negative) can represent a path through the array: every time you "land" on one element in the array, its value tells you which direction to go (is the value positive or negative?) and how many steps to take (the value). Initially, you will start on the first element of the array. There are only three possibilities for ending your walk through the array: you get to the right end of the array, you fall off the left end of the array, or you end up in an infinite loop.

For example, if the array was

0	1	2	3	4
3	3	-1	-1	2

(here, the array values are inside the grid, while the indices are written above the array for reference), then the path would start in position 0 of the array, move 3 steps forward to position 3 of the array, then continue through the positions as follows:

```
start -> 0 -> 3 -> 2 -> 1 -> 4 -> right end
```

Note that if we get to any point where we fall off either end of the array, it is OK to do so by more steps than necessary: in this case, it's OK that position 4 of the array contains a 2, even though that sends us more spaces than we need off the right end of the array.

Write a program which reads in arrays and determines which of the three events happens.

Input

The input for this program consists of several lines of integers. Your program should terminate when it encounters no more input. Each line will consist of at most 40 integers, positive or negative, with one space between each integer. There is no space between the minus sign and a number in the case of negative numbers.

Output

For each line of integers, output either "`left`", "`right`" or "`loop`" depending on which case we are in.

Sample Input	Sample Output
3 3 -1 -1 2	right
3 -1 -1 -1 -1 5 3	loop
4 2 3 5 -10 7 2 5 13 14	left

Judging Data for Problem 3

```
3 3 -1 -1 2
3 -1 -1 -1 -1 5 3
4 2 3 5 -10 7 2 5 13 14
0 1 2 3 4 5 6 7
1 2 3 4 5 6 7
2 0 1
2 0 -1
2 -1 -1
1 1 1
1 -1
1 0
1 1
-1 0
-1 3000
9001
1
0
-1
2 11 14 -8 3 -20 -5 -1 -10 19 8 -19 15 -17 3 -16 -3 -15 19 -4 -4 -1 5 -12 -20 17 -13 -6
6 -6 14 -15 -20 3 3 -17 1 0 -8 2 -19 4 8 14 -15 15 17 -2 -13 14 -20 -10 -2 -10 -4 2 5 -2
13 15 -6 18 19 -3 -17 8 10 13 12 15 15 -1 -4 -16 -20 9 -15 -18 14 -2 -8 -18 11 3 -1 -18
-18 -12 0 -13 -3 16 -9 -3 -19 12 17 5 16 -7 0 -1 -8 -5 -18 -3 -16 9 15 -16 -4 13 3 -14
-13 -16 -9 3 -2 2 18 -19 -20 -12 11 5 -5 14 -19 14 -9 -11 1 -7 7 -11 -18 -2 -9 16 11 -7
-13 2 15 -13 1 -5 14 13 -13 1 -17 8 -11 -10 -4 5 -7 3 -6 -8 16 1 1 8 0 -11 7 11 -19 14
8 -14 0 7 -3 3 -14 13 -17 -19 5 -5 -11 11 5 3 -16 4 -6 -11 -1 17 -16 17 -7 6 2 -20 -5
6 4 -17 -18 -4 18 1 -9 7 7 -13 3 -9 5 -16 -11 -18 5 1 -6 19 1 11 -11 9 -17 4 -4 -10 -11
5 1 -1 13 5 -2 -7 1 2 -3 18 -5 5 1 -12 -8 18 -12 -13 4 16 0 -1 -12 18 9 11 -4 2 -11
-10 8 -6 -5 15 -3 16 -16 -3 13 14 15 7 -3 -8 -8 -13 -7 -6 12 -2 4 -12 5 12 12 3 -20 -15
-18 -18 12 10 -19 -2 6 -9 14 3 -18 19 -15 15 -15 -8 10 -2 13 9 -10 -18 -9 -13 -17 -18
-2 -3 -13 3 8 2 13 5 7 -1 9 -9 3 19 -12 -6 -12 -12 9 9 13 -10 -11 18 4 4 9 -16 17 -18
8 -6 -6 16 11 -8 14 12 -14 17 -4 -12 -8 9 -17 2 -20 -19 0 -3 15 1 -3 19 -20 15 7 17 -6
5 6 -8 4 6 1 -7 -15 14 -2 -18 19 19 -8 -6 -18 14 -19 8 -12 -19 8 10 0 -9 -6 -7 -10 -11 5
7 -1 -7 16 -18 -15 -16 16 -19 -16 14 -18 -5 17 -12 14 -8 16 6 2 -6 -6 8 19 -15 -9 -7 0 14
2 -5 12 -18 -14 10 12 -20 -14 14 -7 -9 9 15 10 -8 -17 -17 -16 -5 -1 -2 -5 -6 1 -4 -2 7
3 4 -9 -19 -13 13 -12 3 6 9 -16 -6 -6 -17 -3 9 -15 -6 -1 0 2 -6 -1 18 8 11 10 0 1 -17
9 10 -8 7 -20 14 -20 18 2 -12 17 11 -1 -18 17 7 10 8 -6 1 -19 -5 7 -17 -9 -15 19 -4 -9 2
8 17 0 13 -6 -3 -2 -13 9 15 6 11 6 -10 5 6 4 12 -18 -6 -19 -10 -8 -10 -7 12 -9 -18 18 -17
2 -1 16 6 -13 15 8 6 5 -16 17 -18 12 -15 -19 12 0 19 -15 5 0 -19 12 16 -8 18 -10 -18 -3
-10 15 -2 8 -8 11 2 0 -8 -14 4 -2 -6 -10 -15 -5 -7 14 18 13 19 -17 -2 -1 9 -16 17 10 -9
-12 -6 -6 -12 1 -7 18 -12 -11 3 -4 -11 -19 -12 -15 17 0 7 -19 -17 15 -1 -6 -11 -17 13 14
3 6 11 -19 12 4 -1 6 15 16 -14 -3 2 -10 -3 14 2 13 19 -8 -15 -19 19 -11 -14 -16 -10 2 -14
-12 -19 5 2 0 -10 6 -4 18 17 -10 9 16 14 3 4 -7 -8 6 -19 -6 18 6 -15 3 -12 -1 7 7 17
12 -11 2 13 15 13 1 4 -11 17 0 -12 -20 0 9 8 9 8 16 12 -3 18 19 -9 8 14 9 0 -14 19
```

Problem 4 – Merging

Write a program which takes two sorted lists of integers and produces a sorted list of all integers in either list, a process which is called merging the lists. If there are duplicates in the lists, then they should be removed: every number in the final list should only appear once.

Input

The first line of the input will contain the number *n* of pairs of lists to be merged. The next *n* pairs of lines contain the sorted arrays to be merged. There are no blank lines in the file. All of the numbers are positive.

Output

For each pair, output the merged list on a single line.

Sample Input	Sample Output
3 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 5 7 9 11 13 1 4 5 5 7 9 4 6 7 10 11 20	1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 7 9 11 13 1 4 5 6 7 9 10 11 20

Judging Data for Problem 4

```
13
1 2 3 4 5
6 7 8 9 10
1 2 3 4 5
5 7 9 11 13
1 4 5 5 7 9
4 6 7 10 11 20
8 12 18 23 25 33 36 36 36 39 43 51 52 61 68 74 82 86 92 98 103 104 112 113 114 114
7 7 9 10 15 15 23 24 33 39 47 48 55 57 65 68 74 74 80 83 91 91 99 102 109 111
0 8 11 17 22 27 29 35 44 50 59 60 62 71 72 74 76 80 83 88 88 88 94 96 102 106
3 12 14 19 26 35 36 36 42 50 53 53 53 55 63 69 78 79 84 90 91 91 91 95 95 95
23 26 29 38 46 52 61 69 70 77 81 87 93 96 97 97 104 111 112 118 124 128 131 137 145 146
6 10 15 17 26 30 37 42 46 54 61 66 68 68 68 70 77 83 83 85 93 102 107 110 119 122
15 16 17 22 31 32 38 38 39 47 48 51 58 58 58 58 60 66 68 72 79 82 84 90 95 104
14 17 17 17 19 20 24 25 30 39 46 46 50 51 56 58 58 61 66 75 77 86 88 92 94 102
23 31 36 43 51 54 62 66 74 82 82 85 86 87 87 90 90 91 98 99 105 107 111 120 121 126
3 11 15 20 21 30 37 45 45 54 54 54 63 69 76 85 94 99 108 117 118 126 128 128 137 139
15 21 22 29 38 39 41 49 49 53 55 59 67 75 76 77 81 89 94 95 98 101 103 104 110 116
20 21 26 33 35 35 37 38 46 48 55 64 68 72 73 76 80 81 85 88 88 92 96 102 105 108
8 11 13 14 23 31 34 38 42 50 50 53 58 63 71 73 81 88 88 91 91 95 96 104 109 110
19 27 36 36 39 44 50 54 58 67 76 84 88 90 93 98 107 111 115 116 118 123 124 130 132 139
12 19 21 23 30 37 37 44 45 49 50 50 56 57 62 65 66 71 72 74 82 87 89 89 98 104
12 19 26 29 35 43 43 51 51 56 65 69 77 83 83 91 96 103 108 108 113 117 121 123 123 126
16 19 26 28 28 37 43 43 44 46 51 59 59 62 66 68 77 80 84 85 89 98 102 104 109 117
16 17 20 28 36 40 42 50 55 55 60 62 62 62 62 66 69 69 74 78 84 91 96 105 105 114
13 14 14 22 22 24 30 30 32 35 35 36 41 47 48 56 59 66 74 74 83 90 90 92 94 96
24 29 38 47 56 62 71 76 83 85 92 98 100 106 108 112 112 115 119 123 126 127 136 141 143
```

Problem 5 – Checkers

Checkers is a well-known board game where you jump over your opponent's players to capture them. Each move is made diagonally, and you begin by moving forward on the board.

In this problem, we ask you to write a program which takes two snapshots of a checkers board, and asks you to verify that one move by one player on the first board can lead to the second board's snapshot.

A snapshot of the board will look like this:

```
_X_X_X_X
X_X_X_X_
___X_X_X
__X_____
_0______
__0_0_0_
_0_0_0_0
0_0_0_0_
```

As you can see, the two teams are represented by Xs and 0s. The blank spaces in the board are represented by underscores (_).

You can make the following assumptions that will simplify your program:

1. X moves down the board, while 0 moves up on the board.
2. It is X's turn.
3. There will always be at least one X and one 0 on the board.
4. No 0s will move unless they are taken from play by move by X.
5. Pieces will not be added to the board.
6. There are no multiple jumps.
7. There is no kinging (so moves always go from top to bottom in the diagram).

Input

The input consists of a first line which contains a single number `n`, the number of test cases. After that, there are `2*n` pairs of 8-by-8 character grids, giving `n` (before,after) pairs that you will test. After each grid is a blank line.

Output

For each case, print out either Valid or Invalid on a line by itself. Print Valid if you can go from the before snapshot to the after snapshot by a valid checkers move. Otherwise, print Invalid.

Sample Input	Sample Output
2 _____ _____ __X_____ _____ _____ _____ _____ _____ _____ _____ _____ ___X____ _____ _____ _____ _____ _____ _____ X_X____ _____ _____ _____ _____ _____ _____ _____ _____ _X_X____ _____ _____ _____ _____	Valid Invalid

Judging Data for Problem 5

For ease of reading, the judging data has been arranged as test cases. Note that the lines "Test case x" do not appear in the test file, and that the beginning of the test file should be the number 14.

Test case 1

```
________
__X_____
________
________
________
________
______0_
________

________
__X_____
________
________
________
________
______0_
________
```

Test case 2

```
________
__X_____
________
________
________
________
______0_
________

________
________
_X______
________
________
________
______0_
________
```

Test case 3

```
________
_X______
__X_____
________
________
_____0__
____0_0_
_____0__

________
________
X_X_____
________
________
_____0__
____0_0_
_____0__
```

Test case 4

```
________
_X______
__X_____
________
________
_____0__
____0_0_
_____0__

________
________
X_______
_X______
________
_____0__
____0_0_
_____0__
```

Test case 5

```
________
_X______
__X_____
________
________
_____0__
0___0_0_
_____0__

________
________
__X_____
________
________
_____0__
0___0_0_
_____0__
```

Test case 6

```
________
_X______
__X_____
________
________
_____0__
____0_0_
_____0__

________
_X______
__X_____
________
________
_____0__
____0_0_
_____0__
```

Test case 7

```
________
_X______
__X_____
________
____0___
_____0__
____0_0_
_0___0__
```

```
________
________
_XX_____
________
____0___
_____0__
____0_0_
_0___0__
```

Test case 8

```
X_______
_X___X__
X_X_____
________
____0___
___0_0__
____0_0_
_0___0__
```

```
X_______
_X______
X_X_____
_______X
____0___
___0_0__
____0_0_
_0___0__
```

Test case 9

```
X_X_X_X_
_X_X_X_X
X_X_X___
_____X__
____0___
_0_X___0
0_0_0_0_
_0_0_0_0
```

```
X_X_X_X_
_X_X_X_X
X_X_X___
________
____0___
_0_X___0
0_0_0_0_
_0_0_0_0
```

Test case 10

```
X_X_X_X_
_X_X_X_X
X_X_X___
_____X__
____0___
_0_____0
0_0_0_0_
_0_0_0_0
```

```
X_X_X_X_
_X_X_X_X
X_X_X___
________
________
_0_X___0
0_0_0_0_
_0_0_0_0
```

Test case 11

```
X_X_X_X_
_X_X_X_X
X_X_X_X_
_____0__
________
_0_____0
0_0_0_0_
_0_0_0_0
```

```
X_X_X_X_
_X_X_X_X
X_X___X_
________
____X___
_0_____0
0_0_0_0_
_0_0_0_0
```

Test case 12

```
X_X_X_X_
_X_X_X_X
X_X_X_X_
_____0__
________
_0_____0
0_0_0_0_
_0_0_0_0
```

```
X_X_X_X_
___X_X_X
X___X_X_
___X_0__
________
_0_____0
0_0_0_0_
_0_0_0_0
```

Test case 13

```
________
_______X
________
________
________
________
__0_____
________

________
________
_______X
________
________
________
__0_____
________
```

Test case 14

```
________
_______X
______0_
________
________
________
__0_____
________

________
________
________
_____X__
________
________
__0_____
________
```

Problem 6 – Downtown Cores

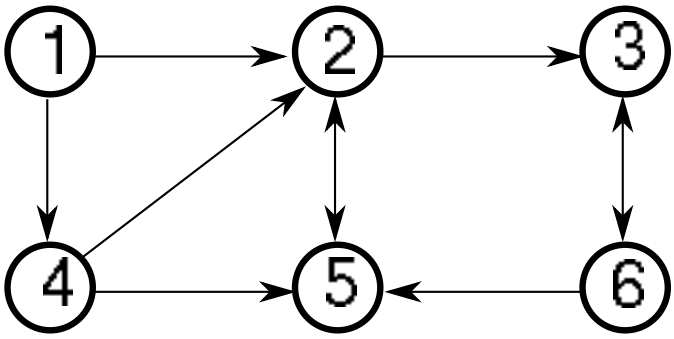
The downtown core of a city has started to be redesigned by making some streets one-way, keeping some two-way, and closing some to all traffic. For each design, the city planners want to know if it is possible for a car to go from a fixed point in the downtown to another point by following the directions of traffic (no going the wrong way up one way streets).

Each intersection in the downtown is given a number, and then sections of road are given in two lists: two-way streets and one-way streets. Each street is given in block-by-block sections, regardless of how long the street is.

For instance, suppose there are six intersections, and the streets are listed as follows:

One way:	Two way:
1 > 2	2 <> 5
2 > 3	3 <> 6
1 > 4	
4 > 2	
6 > 5	
4 > 5	

This would correspond to the following map:



Note that we do not assume that all city blocks are perfect squares. We also do not give a physical location of the intersections in relation to the others: any possible placement of the intersections is suitable to give a solution.

As an example, in this grid, it is possible to travel from intersection 1 to intersection 5 (by going through 2) but it is not possible to travel from 5 to 4.

Input

The first line of input will be the number of downtown layouts your program will test. For each downtown, you will first read (from one line) the number of intersections, then the number of one way segments, followed by the number of

two-way segments. Next, the one-way segments will appear in the form `i > j` (meaning a one-way street from intersection `i` to intersection `j`), followed by the two-way segments, in the form `i <> j` (a two way street between intersections `i` and `j`). For each segment, the integers `i` and `j` will be different.

Following the details of each downtown layout is a single test case, specified by two integers `s` and `t`. This should be interpreted as the query "is it possible to travel from intersection `s` to intersection `t`?" The integers `s` and `t` will be different.

Output

For each downtown layout, output only "`yes`" or "no" on a line,

Sample Input	Sample Output
1 6 6 2 1 > 2 2 > 3 1 > 4 4 > 2 6 > 5 4 > 5 2 <> 5 3 <> 6 5 4	no

Judging Data for Problem 6

```
8
4 4 0
1 > 2
2 > 3
3 > 4
4 > 1
1 4
4 3 1
1 > 2
3 > 4
4 > 1
2 <> 3
3 1
4 3 1
1 > 2
3 > 4
1 > 4
2 <> 3
3 1
4 3 1
2 > 3
3 > 4
1 > 4
1 <> 2
4 2
4 3 0
1 > 2
1 > 3
1 > 4
1 2
4 3 0
1 > 2
1 > 3
1 > 4
3 2
6 6 2
1 > 2
2 > 3
1 > 4
4 > 2
6 > 5
4 > 6
2 <> 5
3 <> 6
5 4
6 6 2
1 > 2
2 > 3
1 > 4
4 > 2
6 > 5
4 > 5
2 <> 5
3 <> 6
1 5
```