Manitoba High School Programming Contest

University of Manitoba

May 28, 2010 12:30-3:30 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 – Hands

As part of a program to visually display an analog clock, you are responsible for converting the time (given in hours:minutes) into two angles representing the positions of each of the two hands.

Angles are measured starting from the 12 o'clock position: 0 degrees would represent a hand pointing up, while 270 degrees would represent a hand pointing to the left, for example.

Your angles should have **at most one decimal digit**. You may assume that each hand moves evenly through its turns: for example, at half past the hour, the hour hand is exactly half way between two hours. The angles should be between 0 and 359 (inclusive).

Suggestion: you shouldn't use any trigonometry (sine, cosine, etc.) for this question. The angles that each hand rotates in one minute should be easily calculated.

Input

The input is of the form `X Y` where each `X`,`Y` is a number. This represents a valid time. The first integer `X` is the hour, a number between `1` and `12`. The second integer `Y` is the minutes, a number between `0` and `59`. There is a single space between the hour and the minute. There are no leading zeroes in the hour, but the minute may have leading zeroes.

Output

Output (on a single line) two numbers representing the angle of the hour and minute hands.

Sample Input	Sample Output
`3 14`	`97.0 84.0`

Judging Data for Problem 1

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

File 1:
4 30

File 2:
10 15

File 3:
11 59

File 4:
12 00

File 5:
12 05

Problem 2 – Ensuring Proper Social Insurance

Canadian Social Insurance Numbers (SINs) have a built-in redundancy code in the form of a check sum. To calculate whether a 9-digit SIN is valid, a calculation is performed. If the calculation arrives at the correct value, the SIN is valid, otherwise it is invalid.

For the 9-digit SIN numbers, the (slightly simplified) calculation is:

1. The digits in even numbered positions are each multiplied by two, and the results are added together.
2. The digits in odd numbered positions are added together.
3. The two sums from steps 1 and 2 are then added together. If this result is evenly divisible by ten, the SIN is valid. Otherwise, the SIN is invalid.

For example, if the SIN is 123456789, then

1. (2*2) + (2*4) + (2*6) + (2*8) = 40
2. 1 + 3 + 5 + 7 + 9 = 35
3. The sum is 75, which is not evenly divisible by 10, so the SIN is invalid.

Input

The first line contains a single value, which is the number of test cases n. After the first line, there are n SINs, each on their own line. Each SIN has nine digits, each separated by a space for ease of reading.

Output

For each case, output either "`valid`" or "`invalid`".

Sample Input	Sample Output
2 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 4	invalid valid

Judging Data for Problem 2

```
9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 4
1 0 1 0 1 0 1 0 6
2 0 2 0 2 0 2 0 2
0 1 0 1 0 1 0 2 0
1 1 1 1 2 1 8 0 1
1 1 1 1 2 1 8 2 0
1 1 1 1 2 1 8 2 9
3 1 4 1 5 2 4 1 4
```

Problem 3 – Risky Cards

Risk ("the game of strategic conquest") is a board game in which players battle each other for military control of the planet, often destroying real-life friendships in the process. The game is complex, but in this problem, we focus on one aspect: redeeming your Risk cards.

After each successful turn, a player is awarded a card. The cards each have a type (in the original Risk these are "infantry", "cavalry" and "artillery", as well as wild cards, which we ignore here). The cards can be redeemed for additional troops, but only when the player has a set. A set is either:

1. three cards of the same type (e.g., three infantry cards).
2. three cards of three different types (e.g., one infantry, one cavalry and one artillery).

Write a program that determines, given a list of cards, whether it has a redeemable set. If several sets can be formed, output them in the following order:

- If a set of the same type can be formed, output it. If no set of the same type can be formed, then output set of three different types (if available).
- If more than one set (same type or different type) can be formed, output the first one that appears in the order they are presented (i.e., if your cards are `X,X,X,Y,Y,Y`, the set you should output 3 Xs as the set).

Input

Each line of input consists of a string of characters, each representing a type of card. For instance "*aabba*" would represent that you had three cards of type "*a*" and two of type "b". Notice that the cards are not sorted and can appear in any order.

Output

If a set of three cards of the same type can be formed, output "`set of three:` *a*" where *a* is the type of the card. If only a set of three different cards can be formed, output "`set:` *abc*" where *a*,*b* and `c` are the three different card types (in the order they appear). If no redeemable set exists in the input string, output "`no set`".

Sample Input	Sample Output
aaa *aab* *aabc*	`set of three:` *a* `no set` `set:` *abc*

Judging Data for Problem 3

```
aaa
aab
abc
aaaa
abbb
ababc
aabbc
aabbb
aabb
aabbc
aabbcc
ababa
ababaaa
abababbb
ababab
abcabcabc
```

Problem 4 – Tapped out.

Jane is required to create solutions of liquid adamantium and water. However, the only tools she has are a stopwatch with a second hand and a tap that she knows pours out 100 mL/s (or 0.1 L/s). Each request for a solution of adamantium and is given by two quantities:

- n: the size of the container that the adamantium is in, in litres. Initially, the container is filled with 100 % liquid adamantium.
- x: the percentage of adamantium in the final solution. The number will be between 1 and 100.

So, if n=10 and x=15, initially Jane gets a 10 L container of adamantium and is required to return the container with 1.5 L of adamantium and 8.5 L of water. To do her job, Jane simply turns on the tap, puts the container underneath the tap, and lets the tap run until she gets the concentration she wants, allowing the mixture to overflow the container in the process. She has made some simplifying assumptions:

- For each second that the liquid overflows the container, 100 mL of water goes in and 100 mL of the liquid leaves the container **at the concentration at the beginning of that second.** This is a simplification, but Jane has figured out her boss never seems to notice.
- Jane might not get the proper concentration at the end of a 1 s interval, but again, her boss never seems to notice. When she can't obtain the proper concentration at the end of a second, she simply over-dilutes the solution and stops at the end of the next second. So if she needs to run the tap for 9.6 s, she will always run it for 10 s.

If her boss ever complains, Jane has rehearsed what she will say: "If you want me to do my job better, maybe you'll want to give me more than just a tap and a stopwatch."

Write a program that will help Jane calculate how long she will run the tap to get the proper concentration.

Input

The input consists of a series of lines, each containing two integers (n,x). n is the size of the container in litres ($n \geq 1$), and x is the target concentration ($1 \leq x \leq 100$). The numbers n and x are separated by a single space. The last line of input is 0 0. This line should not be processed.

Output

For each problem, output "`Run tap for K second(s).`" where K is the solution to the problem. Always output "`second(s)`" – don't worry if it is 1 second. Don't output anything for the line `0 0`.

Sample Input	Sample Output
1 72 2 80 3 100 0 0	Run tap for 4 second(s). Run tap for 5 second(s). Run tap for 0 second(s).

Judging Data for Problem 4

```
10 100
10 99
10 98
10 97
10 96
10 95
10 90
10 80
10 70
10 60
10 50
20 100
20 50
20 1
100 1
1000 1
10000 1
10000 10
10000 20
0 0
```

Problem 5 – Cinderella Distance

In a city with perfectly square city blocks, you are given the average house price on each block (as multiples of $100,000). You are interested in how far it is from the block with the largest average house price to the block with the lowest average house price. The distance is measured in blocks:

- if the blocks are directly next to each other, their distance is 1.
- each additional block between the blocks adds one to their distance.
- only horizontal and vertical moves are allowed to calculate distance: diagonal moves through blocks are not permitted.

You may assume that the largest and smallest average prices each only occur in one unique block in the grid. You may also assume that the two extreme prices occur in two different blocks in the grid.

Input

The first line of the input will be the number of test cases you are to examine. For each test case, the first line gives the width w and depth d (in number of blocks) of the city area that follows. Finally, the next d lines each have w numbers, each of which is an integer and is the average price for houses on that block.

Output

For each case, output "`Distance for case #x: `" followed by the distance, in blocks, as described above. Include one space between the colon and the distance. You do not need a terminating period. The case number x starts at 1.

Sample Input	Sample Output
3 3 5 10 9 18 15 14 25 11 12 13 12 13 14 15 16 17 3 3 1 2 3 4 5 6 7 8 9 2 2 1 5 3 4	Distance for case #1: 2 Distance for case #2: 4 Distance for case #3: 1

Judging Data for Problem 5

```
17
1 2
10
30
2 1
30
10
2 1
10 30
2 1
30 10
2 2
1 2
3 4
2 2
1 4
2 3
2 2
1 3
4 2
2 2
4 1
2 3
2 2
2 1
4 3
2 2
2 1
3 4
2 2
2 3
4 1
2 2
2 4
3 1
2 2
4 2
3 1
```

(continued)

```
3 5
10 9 18
15 14 25
11 12 13
12 13 14
15 16 17
3 3
1 2 3
4 5 6
7 8 9
10 10
6 97 59 44 38 69 27 96 17 90
34 18 52 56 43 83 25 90 93 60
93 14 50 47 8 46 44 9 77 59
16 1 70 77 39 92 71 67 78 51
53 12 19 63 40 90 3 49 49 67
74 90 74 27 98 72 2 73 85 41
4 44 13 19 10 15 64 9 12 52
25 72 90 18 43 55 40 17 70 52
81 87 34 85 9 68 83 63 70 36
36 10 40 66 87 16 92 43 53 61
10 5
47 28 75 33 5 11 37 75 4 91
22 40 58 93 98 11 30 6 32 40
24 80 96 11 23 41 52 58 67 81
65 69 25 36 61 84 96 94 31 81
31 99 67 59 66 12 49 90 35 15
```

Problem 6 – Family Forest?

You are hired by a company to assemble family trees for clients based on information on relationships. For confidentiality reasons, you are not given names of individuals, but each person is represented by a unique 8-digit number. The information provided to you is of the form: XXXXXXXX P YYYYYYYY or XXXXXXXX S YYYYYYYY where XXXXXXXX and YYYYYYYY are (maximum) 8-digit numbers and P and S represent "parent" and "spouse". As an example, the information

4 S 5

represents that individual 4 is the spouse of individual 5.

You are responsible for constructing a family tree for one individual (the "target"), and reporting whether the tree is

1. **incomplete**, due to an insufficient amount of parent/child or spouse information. This occurs when an individual listed who is not the target does not have any children listed in the family tree (that is, the information does not make up a single tree, but a bunch of unconnected, separate trees.)
2. **contradictory**, because of too many parents (maximum 2 parents/individual). So if either (a) one individual has more than two parents listed, or (b) an individual has more than one spouse listed, then the entire tree is contradictory.
3. **acceptable**. neither of (a) or (b) occurs.

Notice that we don't care about the gender of individuals, or single-parent families (or when we only have information about one parent). We also don't care about sibling relationships. Finally, we assume that there are no cycles in the tree (no one gives birth to their own grandparent).

If a set of input is both contradictory and incomplete, it is listed as contradictory.

Input

The first line contains a single value, which is the number of test cases n. After the first line, there are n cases. Each case begins with an integer k on a single line. The next line contains the ID integer of the target individual. The next k lines are all relationships of the form described above.

Output

For each test case, output one of the words "`incomplete`", "`contradictory`" or "`acceptable`" on a line.

Sample Input	Sample Output
3 2 1 2 P 1 4 P 3 3 1 2 P 1 3 S 2 4 S 3 4 2 1 P 2 3 P 2 4 P 3 5 S 4	incomplete contradictory acceptable

Judging Data for Problem 6

```
13
2
1
2 P 1
4 P 3
3
1
2 P 1
3 S 2
4 S 3
4
1
2 P 1
3 P 1
4 P 3
5 S 4
4
3
1 P 2
2 P 3
4 P 5
5 P 3
3
2
1 P 2
3 P 2
4 P 2
9
1
2 P 1
3 S 2
4 P 2
5 P 2
6 P 3
7 P 3
9 P 6
8 S 9
10 P 7
```

(continued)

```
9
1
2 P 1
3 S 2
4 P 2
5 P 2
6 P 3
7 P 3
8 S 9
9 P 6
10 P 7
8
1
10 P 7
9 P 6
8 P 6
7 P 3
6 P 3
3 P 1
2 P 1
5 P 2
6
7
1 P 5
5 P 7
6 P 7
4 P 6
2 P 5
3 P 5
7
7
1 P 4
2 P 4
4 P 7
5 P 7
3 P 5
6 S 3
8 S 6
```

(continued)

```
7
7
1 P 4
2 P 4
4 P 7
5 P 7
3 P 5
6 S 3
8 S 3
3
4
1 S 2
3 P 4
1 P 4
4
1
2 P 1
3 P 1
4 P 1
5 P 6
```