University Interscholastic League

Computer Science Competition

2005 Regional Programming Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 10.

2. All problems have a value of 72 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Point Values and Names of Problems

Number	**Name**	**Point Value**
Problem 1	Assembly Required	72
Problem 2	Brain Plan	72
Problem 3	Copier Crime	72
Problem 4	What's the Date?	72
Problem 5	Dig Dog	72
Problem 6	To The Death	72
Problem 7	As the Gear Turns	72
Problem 8	Juggling Numbers	72
Problem 9	Speeding Takes its Toll	72
Problem 10	What's the Word?	72
Total		**720**

Problem 1 **72 Points**

Assembly Required

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

You just purchased a brand new desk, but the problem is that it's in pieces and requires assembly. You begin by removing all of the hardware, most of which consists of several sets of screws of different sizes. You read through the instruction manual to find that each step refers to the required screw by its size. The problem is that you have no idea of knowing the size of the screws since they are not labeled (and you don't have a ruler).

You come up with a clever idea that involves comparing the screws based on their size (you put screws next to each other to determine which is longer). This allows you to arrive at an ordering of screws from longest to shortest. Since you know the expected sizes, you can then determine which screws are which length and, correspondingly, which screws need to be used for the current step. This process turns out to be tedious, so you decide to write a program to figure it all out for you.

Note that different types of screws are always different lengths.

Input

The first line will contain a single integer *n* indicating the number of data sets.

The first line of each dataset will contain a single integer $m < 10$ indicating the number of sets of screws, which are labeled as 'A', 'B', 'C', etc.

Each of the next *m* lines form a matrix, like a multiplication table, relating the sizes of screws. The first line compares set 'A' to each of the other sets using the characters '<', '>', and '=' to indicate that its length is less than, greater than, or equal to the length of screws in the corresponding column. The next line holds the same information for set 'B', and the pattern continues for each of the other types of screws. Note that since all screw types are of different sizes, the '=' will only appear along the main diagonal of the matrix.

The next line in each dataset will list the screw lengths, in descending order, as they are listed in the manual.

The last line in each dataset will consist of an integer that corresponds to the length of the screw required for the next step.

Output

For each dataset, display the string "SET *X*" where *X* is the character label of the set where you get the next required screw.

Example Input File

```
2
5
= > > > >
< = > > >
< < = > >
< < < = >
< < < < =
5 4 3 2 1
1
3
= > <
< = <
> > =
15 5 1
5
```

Example Output To Screen

```
SET E
SET A
```

Problem 2 **72 Points**

Brain Plan

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

Researchers are developing non-invasive devices that allow patients to control robotic arms using their minds. These devices examine the brainwave readings in patients to determine what action the robotic arm should take. Before the device can function, it needs to be programmed to associate certain brainwave patterns with robotic arm movements. To aid in this effort, two test subjects have had their brainwave readings taken when trying to get the robotic arm to perform specific actions.

Your job is to generalize the sets of brainwave readings into brainwave patterns that the device can use for comparisons in upcoming trials. Brainwave scans from the test subjects show only active and inactive portions of the brain. From these scans, a pattern can be deduced by determining where the scans agree and where they disagree. Areas of agreement indicate portions of the pattern that should be active (or inactive) while disagreements indicate portions of the pattern that should be marked as unimportant.

Brainwave scans are strings of 18 characters where each character represents a portion of the brain that is either 'A'=Active or 'I'=Inactive. Patterns are also strings of 18 characters where 'A'=Active, 'I'=Inactive, and '*'=Unimportant.

For instance, the following pair of brainwave scans:

```
AAAAIIIIIIIIIAAIAI
IIAAIIIIAIIIIAIIAI
```

gives rise to the pattern:

```
**AAIIII*IIIIA*IAI
```

Input

The first line will contain a single integer *n* indicating the number brainwave scans pairs that need to have their patterns calculated. Each pair of the next 2*n* lines will contain brainwave scans for different actions.

Output

For each brainwave scan pair in the input, output the corresponding brainwave pattern on its own line.

Example Input File

```
3
AAAAIIIIIIIIIAAIAI
IIAAIIIIAIIIIAIIAI
AAAAAAAAAIIIIIIIII
IIIIIIIIIAAAAAAAAA
AIAIAIAIAAIAIAIAIA
IAIAIAIAIAIAAAIAAI
```

Example Output To Screen

```
**AAIIII*IIIIA*IAI
******************
*********AIA*AIA**
```

Problem 3 **72 Points**

Copier Crime

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

Though few people know it, many manufacturers of color laser printers and color copiers encode identifying information onto every sheet of paper that passes through their machines, enabling counterfeit bills to be traced back to the machines that printed them.

In this case, assume that the identifying marks are a series of small yellow dots and that you, as a government agent tasked with catching a counterfeiter, need to determine the serial number of the machine that made some counterfeit bills. The series of dots have already been transcribed for you, with dashes filled in to denote blank space. Each dot/dash sequence is fifteen units long, representing a series of five, three-digit binary numbers (with a dot representing '1' and a dash representing '0'). A decoded serial number is constructed by concatenating the decimal equivalents of each of the binary triplets.

For instance, the sequence:

```
..-.--..-...-.-
```

represents:

```
110100110111010
```

which broken into triplets is:

```
110 100 110 111 010
```

translating each triplet to decimal gives:

```
 6   4   6   7   2
```

So the serial number represented is:

```
64672
```

Your task is to decode each dot/dash sequence and display the corresponding serial number.

Input

The first line will contain a single integer *n* indicating the number of serial numbers that need to be translated. Each of the next *n* lines will contain a fifteen character sequence of dots (periods) and dashes (hyphens).

Output

For each encoded serial number listed in the input, display the decoded serial number on its own line.

Example Input File

```
4
..-.--..-...-.-
...---...---...
---...---...---
-----.-.--...--
```

Example Output To Screen

```
64672
70707
07070
01234
```

Problem 4 **72 Points**

What's the Date?

Program Name: date.java **Input File: date.in**

Given the current date and the number of days until a person turns 18, display the date of that person's 18th birthday.

To do this, you'll need to know the number of days in each month. Since February only has a 29th day on leap years, you also need to know the algorithm for determining whether or not a given year is a leap year.

Number of days in each month:

January	31
February	28 (29 on a leap year)
March	31
April	30
May	31
June	30
July	31
August	31
September	30
October	31
November	30
December	31

A year is a leap year if it is either: a) evenly divisible by 400 or b) divisible by 4 and **not** divisible by 100. For example, 1976 is a leap year because 4 divides it and 100 doesn't. 1900 is not a leap year because 400 doesn't divide it and 100 does, and 2000 is a leap year because 400 divides it.

Input

The first line will contain a single integer *n* indicating the number of birthdays that will need to be calculated. Each of the next *n* lines will contain a first name, the long form of a start date (i.e., <month> <day>, <year>), and a positive integer representing the number of days between the start date and the date the person turns 18.

All input dates will fall between years 1900 and 2100, and the number of days will always be less than 10,000.

Output

For each person listed in the input, display on a single line a sentence declaring the date of the person's 18th birthday by printing, "`<name>'s 18th birthday is on <month> <day>, <year>`".

Example Input File

```
3
Bob January 23, 2005 365
James January 1, 1990 1662
Tim May 18, 2003 1
```

Example Output To Screen

```
Bob's 18th birthday is on January 23, 2006
James's 18th birthday is on July 21, 1994
Tim's 18th birthday is on May 19, 2003
```

Problem 5 **72 Points**

Dig Dog

Program Name: digdog.java **Input File: digdog.in**

Bailey is a bad dog sometimes. Sometimes she doesn't always dig where her owner likes. Given a layout of the owner's yard, determine if Bailey has dug in a spot that would make the owner angry.

Input

The first line will contain a single integer *n* indicating the number of datasets. Each dataset will consist of 3 components:

1. The first line will contain a single integer *m*, representing the length and width of the yard.
2. The next *m* lines represent the layout of the yard, where each line contains *m* characters. A numeric character represents a spot that would make the owner angry if Bailey dug there. Acceptable digging spots are represented by periods ('.').
3. The next *m* lines represent the layout of the yard, where each line contains *m* characters. An "X" character represents a spot where Bailey dug.

Output

For each data set, if Bailey has dug in a spot that would make the owner angry, display the string "BAD DOG". Otherwise, display the string "GOOD DOG".

Example Input File

```
4
5
...1.
.2.1.
.22..
..2..
...3.
.....
..X..
X..XX
.....
...X.
3
.45
9..
..6
X..
.XX
XX.
2
12
34
..
..
2
12
34
XX
XX
```

Example Output To Screen

```
BAD DOG
GOOD DOG
GOOD DOG
BAD DOG
```

Problem 6 **72 Points**

To The Death

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

You are programming the battle engine for an upcoming role-playing game (RPG). Your program is given the hit points (HP), weapon strength, and armor strength of the player and the monster that player is engaging in battle. Your battle engine is to run the battle until either the player or the monster has been slain and announce the victor of the battle.

The battles are run in a turn-based fashion, and the player always gets the first attack. The attack power of each fighter is calculated by adding that fighter's HP and weapon strength, and the defense power of each fighter is calculated by adding that fighter's HP and armor strength. Damage is determined by taking the attacker's attack strength and deducting from it the defense power of the fighter being attacked; if the damage calculated is less than or equal to zero, the damage is instead equal to 10% of the attacker's attack power (rounded up). The amount of damage is deducted from the HP of the fighter being attacked. During each round of the battle both fighters take turns attacking and defending. A fighter is slain if his HP reaches or falls below zero.

Input

The first line will contain a single integer *n* indicating the number of data sets.

The first line in each dataset will consist of 3 positive integers that correspond to the player's HP, weapon strength, and armor strength, respectively.

The last line in each dataset will consist of 3 positive integers that correspond to the monster's HP, weapon strength, and armor strength, respectively.

Output

For each dataset, print the following:

"BATTLE *X*" (where *X* is the number of the dataset)

For each attack, display the damage scored as follows:
"The player attacks the monster for *x* damage!" or "The monster attacks the player for *x* damage!"
Where *x* is the damage scored for that turn.

If the player is slain, display the string "The player has been slain!".
If the monster has been slain, display the string "The monster has been slain!".

Example Input File

```
2
50 50 35
50 40 30
35 25 40
50 30 60
```

Example Output To Screen

```
BATTLE 1
The player attacks the monster for 20 damage!
The monster attacks the player for 7 damage!
The player attacks the monster for 33 damage!
The monster has been slain!
BATTLE 2
The player attacks the monster for 6 damage!
The monster attacks the player for 8 damage!
The player attacks the monster for 6 damage!
The monster attacks the player for 1 damage!
The player attacks the monster for 6 damage!
The monster attacks the player for 7 damage!
The player attacks the monster for 5 damage!
The monster attacks the player for 6 damage!
The player attacks the monster for 4 damage!
The monster attacks the player for 6 damage!
The player attacks the monster for 4 damage!
The monster attacks the player for 2 damage!
The player attacks the monster for 3 damage!
The monster attacks the player for 1 damage!
The player attacks the monster for 3 damage!
The monster attacks the player for 5 damage!
The player has been slain!
```

Problem 7 **72 Points**

As the Gear Turns

Program Name: gear.java **Input File: gear.in**

Gears are numbered 1 to 9. If a gear turns in the clockwise direction, the gears to the left and right of it turn counter-clockwise. Conversely, if a gear turns in the counter-clockwise direction, the gears to the left and right of it turn clockwise.

Input

The first line will contain a single integer *n* indicating the number of datasets. Each dataset will consist of 2 components:

4. The first line will contain a sequence of nine gears, separated by spaces. The gears will be numbered from 1 to 9.
5. The next line contains the direction ("CLOCKWISE" or "COUNTER-CLOCKWISE") of gear 1.

Output

For each dataset, display "CLOCKWISE" or "COUNTER-CLOCKWISE" depending on the direction of gear 9.

Example Input File

```
3
3 2 9 1 6 5 4 7 8
CLOCKWISE
4 6 2 1 8 9 3 5 7
CLOCKWISE
1 2 3 7 8 9 6 5 4
COUNTER-CLOCKWISE
```

Example Output To Screen

```
COUNTER-CLOCKWISE
CLOCKWISE
CLOCKWISE
```

Problem 8 **72 Points**

Juggling Numbers

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

Did you ever wonder how jugglers can keep track of all of those balls while not letting any of them fall? They do it by mastering a certain number of basic throws and then chaining them together in exciting ways.

One convenient side-effect of this technique is that it is possible to represent a juggling pattern fairly well with a string of single digits such as “5313” or “441441”. Each digit represents a single throw, with the height of the throw corresponding to the size of the number (the exception is the 0 digit, which represents no ball is thrown during that step). For instance, a ball thrown with a height of 5 will have to be caught five steps later in the sequence. Not all sequences are possible for jugglers, however, since they can only catch one ball during any given step. In the sequence “321” all three balls would be landing during step 4! It’s very useful to be able to determine whether or not a sequence is possible.

Input

The first line of input will contain a single integer *n* indicating the number of datasets.

The following *n* lines will each contain a sequence of digits (0-9) representing a juggling pattern. Each sequence will contain from 1 to 40 digits.

Output

For each dataset in the input, determine if there is any step where more than one ball must be caught. If there is no such step, then the sequence is valid and the string “VALID” should be displayed. Otherwise, for sequences with steps where multiple balls have to be caught simultaneously, the sequence is invalid, and we want to know the number of balls that will drop the first time the juggler misses. This value should be one less than the total number of balls that need to be caught during the first step where the juggler has to catch more than one. In the case of an invalid sequence, display “DROPPED *X* on step *Y*”, where the first drop occurs on step *Y*, with *X* balls missed, and steps are numbered starting at 1.

Note the sequences will likely end with some balls still in the air. Solutions should treat this situation as if the sequence ended in just enough zeros (“0”) to ensure all balls were caught.

Example Input File

```
4
333333333
441441441441
333321
445441441441
```

Example Output To Screen

```
VALID
VALID
DROPPED 2 on step 7
DROPPED 1 on step 8
```

Problem 9 **72 Points**

Speeding Takes its Toll

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

A man receives a toll booth ticket as he drives onto a toll road. Two hours later, he hands the ticket to the toll booth operator as he exits the toll road. She examines the timestamp on the toll booth ticket and, knowing how far the man traveled, promptly issues him a speeding ticket. She explains to the man: “Sir, the speed limit on this toll road is 55 miles per hour. Since you traveled 120 miles in two hours, I can prove mathematically that at some point during your trip, you were going faster than 55 miles per hour. How do I do this? Your average speed is 60mph, which is greater than 55 miles per hour.”

Given this information for other drivers, determine if they deserve speeding tickets as well.

Input

The first line will contain a single integer *n* indicating the number of drivers on the toll road. Each of the next *n* lines will contain a positive integer representing the distance in miles that the driver traveled and a positive integer representing how many hours it took the driver to travel that far.

Output

For each driver, display the string “SPEEDING TICKET” if the driver’s average speed is more than the toll road’s posted speed limit, 55 miles per hour. If the driver’s average speed is equal to or less than 55 miles per hour, display the string “NO SPEEDING TICKET”.

Example Input File

```
4
120 2
190 3
55 1
200 5
```

Example Output To Screen

```
SPEEDING TICKET
SPEEDING TICKET
NO SPEEDING TICKET
NO SPEEDING TICKET
```

Problem 10 **72 Points**

What's the Word?

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

Given a sentence, determine if the first letter in each word spells the first word in the sentence.

Input

The first line will contain a single integer *n* indicating the number of following lines containing sentences to be checked.

Output

For each sentence, if the first letter in each word spells the first word in the sentence display the string "YES". Otherwise, display the string "NO". Consider punctuation marks no differently than alphabetic characters. Also, disregard the case (upper vs. lower) of the alphabetic characters.

Example Input File

```
4
The horse eats.
They have eight young children.
Kimberly is Mr. Belvedere's eighth refined ladylike youngster.
This hour's over.
```

Example Output To Screen

```
YES
NO
YES
NO
```