Computer Science Competition

2010 Cypress Woods 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 12.

2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

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. Names of Problems

Number	Name
Problem 1	Brake Fluid
Problem 2	Tick-Tock Clocks
Problem 3	Baking Cupcakes
Problem 4	D-Day
Problem 5	Food Service
Problem 6	Continued Fractions
Problem 7	Pi
Problem 8	PONG
Problem 9	Even Prime Number
Problem 10	Recall
Problem 11	A Roman Holiday
Problem 12	Store Window

1. Brake Fluid

Program Name: Brake.java **Input File: brake.dat**

Daniel Fredrickson drives a very old truck that leaks brake fluid. Given the level of brake fluid, determine if he needs to refill his brake fluid immediately or if he can wait to refill. If the level of brake fluid is below twenty-three, then it is too low and he needs to refill.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set is composed of a single integer, which is the level of brake fluid.

Output

For each data set, print, `"You need to refill the brake fluid."` if his level of brake fluid is below twenty-three; otherwise, print, `"You will be fine for a little while."`

Example Input File

```
4
35
12
23
6
```

Example Output to Screen

```
You will be fine for a little while.
You need to refill the brake fluid.
You will be fine for a little while.
You need to refill the brake fluid.
```

2. Tick-Tock Clocks

Program Name: Clocks.java **Input File: clocks.dat**

All of your clocks are broken, and each clock is set to a specific date and time with a specific speed. Given the starting date, time, and rate of change of each clock, find the first time at which they intersect. A broken clock can move forwards, backwards, or stand still. All clocks are in military time; that is to say all clocks measure hours in 24-hour intervals, as opposed to two 12-hour intervals.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with a single integer `x` denoting how many clocks are in each set. There will be `x` sets of seven integers to follow (`A, B, C, D, E, F,` and `G)`, where each set pertains to a different clock.

- `A` is the current year of the clock, where $1900 \leq A \leq 2100$
- `B` is the current month , where $1 \leq B \leq 12$
- `C` is the current day of the month, where $1 \leq C \leq 31$
- `D` is the current hour, where $0 \leq D \leq 23$
- `E` is the current minute, where $0 \leq E \leq 59$
- `F` is the current second, where $0 \leq F \leq 59$
- `G` is the number of seconds that pass on the broken clock during one second of real time, where $-1000 \leq G \leq 1000$ (A negative number denotes a clock moving backwards)

Output

Output the exact date and time when all the clocks will first intersect in the following format:
`"M DD, YYYY hh:mm:ss",` where `M` is the full name of the month, `DD` is the day of the month, `YYYY` is the year, `hh` is the hour, `mm` is the minute, and `ss` is the second. There will always be an intersection.

Example Input File

```
2
3
2010 3 1 1 0 0 2
2010 3 1 2 0 0 1
2010 3 1 0 0 0 3
2
2012 2 1 0 0 0 -60
2011 5 3 12 0 0 60
```

Example Output to Screen

```
March 01, 2010 03:00:00
September 17, 2011 06:30:00
```

3. Baking Cupcakes

Program Name: Cupcakes.java **Input File: cupcakes.dat**

Alex is trying to make his favorite cupcakes. In order to do so, he needs to measure out cups of sugar. But some bears have ransacked his house, and all he can find are three tins with no scale markings! He needs help figuring out the steps it will take to balance all of his sugar to make cupcakes. Alex is afraid the bears could come back at any second though, so he needs to balance his sugar using the least number of steps possible. You will write a program that will output the steps it takes to get his sugar tin setup to the correct setup for baking cupcakes. Alex starts off knowing the size of his tins, the amount of sugar he begins with, and the amount of cups of sugar a batch of cupcakes will take to bake (the target amount). His goal is to redistribute all the sugar amongst his tins to make full batches of cupcakes; the number of batches he ends up being able to bake is irrelevant, and not all three tins will necessarily end up with sugar in them. In order to correctly balance his sugar to the target amount, Alex will need to repeatedly pour the sugar from one tin into another in the right order. When pouring from a tin, `J`, to a tin, `K`, he must pour either until `J` is empty or until `K` is full. His setup will be correct when every tin has either the target amount of sugar or no sugar at all.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will contain four integers: `A, B, C, D.` `A` is the size in cups of the first tin, which is full to the brim with all of Alex's sugar. `B` and `C` are the sizes in cups of the other two tins that Alex can use. `D` is the amount of sugar in cups one batch of cupcakes requires. All input integers will be between 1 and 50, inclusive.

Output

The output for each data set will be an indefinite number of lines. The lines will state the shortest amount of steps it takes to balance the sugar in the format, `"Pour the` *`A`* `tin into the` *`B`* `tin (`*`X`* `–` *`Y`* `–` *`Z`*`)"`, where `A` is the size of the tin Alex is pouring from, `B` is the size of the tin Alex is pouring into, and `X`, `Y`, and `Z` are the amounts of sugar in each tin after the transfer of sugar is complete. The last line of output for each data set states the number of steps it took to reach the correct tin setup in the format, `"It took` *`S`* `step(s).",` where `S` is the number of steps. There will only be one shortest solution. There will be one line between each data set.

Example Input File

```
2
8 5 3 4
10 7 3 5
```

Example Output to Screen

```
Pour the 8 tin into the 5 tin (3 - 5 - 0)
Pour the 5 tin into the 3 tin (3 - 2 - 3)
Pour the 3 tin into the 8 tin (6 - 2 - 0)
Pour the 5 tin into the 3 tin (6 - 0 - 2)
Pour the 8 tin into the 5 tin (1 - 5 - 2)
Pour the 5 tin into the 3 tin (1 - 4 - 3)
Pour the 3 tin into the 8 tin (4 - 4 - 0)
It took 7 step(s).

Pour the 10 tin into the 7 tin (3 - 7 - 0)
Pour the 7 tin into the 3 tin (3 - 4 - 3)
Pour the 3 tin into the 10 tin (6 - 4 - 0)
Pour the 7 tin into the 3 tin (6 - 1 - 3)
Pour the 3 tin into the 10 tin (9 - 1 - 0)
Pour the 7 tin into the 3 tin (9 - 0 - 1)
Pour the 10 tin into the 7 tin (2 - 7 - 1)
Pour the 7 tin into the 3 tin (2 - 5 - 3)
Pour the 3 tin into the 10 tin (5 - 5 - 0)
It took 9 step(s).
```

4. D-Day

Program Name: Dday.java **Input File: dday.dat**

General George S. Patton has just reached the Rhine River, forty days into Operation Overlord. Unfortunately, his tank division has completely outstripped his support staff. Impatient, the general has ordered Lieutenant Roger Winters to personally encipher his communication using a rail fence cipher. This particular cipher hasn't been in use since the days of Caligula, so the Lieutenant needs your help to encipher his messages.

When using the rail fence cipher, the plaintext - a.k.a. the original message - is laid out in a zig-zag pattern along "rails" of a "fence." The plaintext starts from the top/left, moves diagonally down and right until it hits the bottom rail, moves diagonally up and right until it hits the top rail, and continues to switch vertical directions until there are no more characters left in the message. To form the enciphered message, read each rail (starting at the top of the fence) in order from left to right, ignoring the space between the letters. The following example illustrates how to rail-encipher the message, "THE TRUTH SHALL SET YOU FREE," using three rails and an output width of five.

```
THE TRUTH SHALL SET YOU FREE 3 5

T . . . R . . . S . . . L . . . Y . . . R . .
. H . T . U . H . H . L . S . T . O . F . E .
. . E . . . T . . . A . . . E . . . U . . . E

TRSLY
RHTUH
HLSTO
FEETA
EUE
```

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. A set of data will include the message to be enciphered, followed by the integers `a` and `b`. `a` will be the number of rails to be used and `b` will be the output width of the enciphered message. `a` and `b` will always be single digit, non-negative numbers.

Output
For each data set, print the enciphered message split up into one or more lines of length `b` followed by a blank line. Any punctuation and spacing in the plaintext should be ignored, and the output should consist of only capital letters and/or numbers.

Example Input File
```
3
THE TRUTH SHALL SET YOU FREE 3 5
Omaha Beach Taken at 20:30 5 3
Have Encountered Resistance in the Hedgerows 3 4
```

Example Output to Screen

```
TRSLY
RHTUH
HLSTO
FEETA
EUE

OCT
MAH
A2A
ETN
0HB
AE3
AK0

HEUR
ETEH
DOAE
NONE
ERSS
ACIT
EEGR
WVCT
DINN
HES
```

5. Food Service

Program Name: Food.java **Input File: food.dat**

Mitch works the front counter at a local fast food restaurant. To appear more personal, he keeps a list of the regular customers and what they like to order. He needs a program that will automatically ring up an order based on the name of a regular customer. Write a program that will output the order of a customer based on a given list of names and usual orders.

Input

The first line will contain two integers, `n` and `k`. The next `n` lines will each contain the name of a regular customer and his/her usual order. The name and the order will be separated by a `"|"`. The next `k` lines will contain the names of all the customers that have entered the restaurant.

Output

For each regular customer that has entered the restaurant, print the customer's usual order in the form `"Customer R ordered F."`, where `R` is the name of the regular customer and `F` is their order.

Example Input File

```
5 3
Jay Jenkins|Whataburger with cheese
Ray Robinson|Number 2 with cheese and bacon
Gerry Geraldo|Chili con carne
Fred Johnson|Chicken strips
Abel|Fries
Ray Robinson
Fred Johnson
Gerry Geraldo
```

Example Output to Screen

```
Customer Ray Robinson ordered Number 2 with cheese and bacon.
Customer Fred Johnson ordered Chicken strips.
Customer Gerry Geraldo ordered Chili con carne.
```

6. Continued Fractions

Program Name: Fractions.java **Input File: fractions.dat**

A continued fraction is a mathematically "pure" method of expressing a number, either rational or irrational, using a set of integers. Continued fractions are infinite when created from irrational numbers and finite when created from rational numbers; for this problem, you only need to worry about real, rational, non-negative numbers. A continued fraction for number *x* is shown below:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \ddots}}}}$$

The simplified notation for number *x* would be $[a_0;\, a_1, a_2, a_3, a_4\ldots]$,
where $x = a_0 + 1 / (a_1 + 1 / (a_2 + 1 / (a_3 + 1 / (a_4 + \ldots))))$.

For example, the rational number 27/8 or 3.375, can be represented in simplified continued fraction notation as [3; 2, 1, 2] because 27/8 = **3** + 1 / (**2** + 1 / (**1** + 1 / **2**)).

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. The following `n` lines will each consist of one fraction: two whole numbers (a numerator and a denominator) separated by a slash `"/"`. Both the numerator and the denominator will not exceed 2^{63}-1.

Output
For each data set, print out the equation for the continued fraction; that is, print the original fraction, a space, an equals sign, another space, and the simplified continued fraction notation for the input fraction. Using the above example, the output would be `"27/8 = [3; 2, 1, 2]"`.

Note: Doubles are inherently inaccurate and cannot represent every decimal value.

Example Input File
```
6
27/8
16/19
111/19
649/200
1/1
355/113
```

Example Output to Screen
```
27/8 = [3; 2, 1, 2]
16/19 = [0; 1, 5, 3]
111/19 = [5; 1, 5, 3]
649/200 = [3; 4, 12, 4]
1/1 = [1]
355/113 = [3; 7, 16]
```

7. Pi

Program Name: Pi.java **Input File: none**

Draw an ASCII art representation of pi using its first twenty-six digits.

Input
None

Output
Print out pi exactly as shown.

Example Output to Screen

```
**3.14159
26535897
**9***3
*2****3
*8****4*6
26****434
```

8. PONG

Program Name: Pong.java **Input File: pong.dat**

You're in the middle of writing code for the game PONG. Your goal for this step of the process is to determine the winner of the game given the initial positions of the paddles and ball and the initial angle of the ball's trajectory. For the sake of this problem, the paddles do not move, are one-dimensional, and are flat up against their respective sides; this means that the paddles are basically just small sections of the left and right walls (not free-floating like in traditional pong). The ball itself is one pixel in size. The dimensions of the entire game field are 600 x 400 (width x height) pixels. The length of each paddle is 75 pixels.

The ball can be in motion for multiple turns. If it collides with the top or bottom wall, it will continue moving in the same x-direction, but the opposite y-direction. If the ball collides with a paddle, it will continue moving in the same y-direction, but the opposite x-direction. When the ball ricochets off of a surface, its angle is reflected vertically or horizontally when hitting a horizontal or vertical surface, respectively. Speed is irrelevant in this problem. When the ball hits a section of the left or right wall that is not a paddle, the game ends, and whichever side the ball hits is the losing side.

Position (0, 0) is the top left corner, and position (599, 399) is the bottom right corner.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. The following `n` lines will consist of four integers: `A`, `B`, `C`, and `D`, followed by a decimal `E`.

- `A` is the y-position of the top end of the left paddle, where $0 \le A \le 324$
- `B` is the y-position of the top end of the right paddle, where $0 \le B \le 324$
- `C` is the x-position of the ball, where $0 \le C \le 599$
- `D` is the y-position of the ball, where $0 \le D \le 399$
- `E` is the initial angle of the ball's trajectory in degrees, where $0.0 \le E \le 359.9$. The direction of the angle is based on the unit circle; 0 degrees is straight right, 90 is straight up, 180 is straight left, and 270 is straight down.

Output

For each data set, print out, `"The _____ wins after` *`X`* `turn(s)."`, where the blank is filled with either `"left"` or `"right"` and the `X` is replaced by the number of times the ball hit a paddle in the game. There will always be a winner.

Example Input File

```
4
155 131 423 296 217.4
48 296 265 225 325.7
153 0 423 296 163.1
158 182 423 296 27.5
```

Example Output to Screen

```
The left wins after 1 turn(s).
The left wins after 6 turn(s).
The left wins after 3 turn(s).
The right wins after 1 turn(s).
```

9. Even Prime Number

Program Name: Prime.java **Input File: prime.dat**

Given a positive non-zero integer; output whether or not it is an even prime number.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of a single integer `d`, where $0 < d < 2^{63}$.

Output
If it is an even prime number, output `"It is an even-prime."`, otherwise output `"This is not what you want."`

Example Input File
```
3
3
2
976532
```

Example Output to Screen
```
This is not what you want.
It is an even-prime.
This is not what you want.
```

10. Recall

Program Name: Recall.java **Input File: recall.dat**

Congratulations! You just inherited a Rental Car company from a distant relative you didn't even know existed. However, many of the cars in your company's inventory have been recalled due to safety concerns. As the new owner of the company, you must go through all of the cars and designate which ones are safe to drive and which ones have to be recalled. If a car was manufactured by Toyota, it must be recalled. Otherwise, it is considered safe to drive.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. The first line of each data set will contain a single integer `c` indicating the number of cars that will follow. The following `c` lines are cars recorded in the format `"Model, Manufacturer"`.

Output

For each set output two lines. The first line should say `"The following models are safe: "` followed by a comma-separated list of the safe car models. If no cars are safe, print `"None"`. The second line should say `"The following models are not safe: "` followed by a comma-separated list of the unsafe car models. If no cars are unsafe print `"None"`. There should be an empty line after the output for each data set.

Example Input File

```
2
3
Mustang, Ford
Prius, Toyota
H3, Hummer
2
Corolla, Toyota
Land Cruiser, Toyota
```

Example Output to Screen

```
The following models are safe: Mustang, H3
The following models are not safe: Prius

The following models are safe: NONE
The following models are not safe: Corolla, Land Cruiser
```

11. A Roman Holiday

Program Name: Roman.java **Input File: roman.dat**

The year is 6 CE, and the Roman emperor Augustus has been toiling in the imperial mansion for twelve years. Finally, his much aggrieved wife, Livia, suggests he take a holiday. Augustus sets forth from Rome and decides to dismiss most of his staff for privacy. Upon arriving in Naples, he realizes to his horror that he has also left his personal accountant behind! In the bread market, he finds himself unable to perform simple fractional subtraction and addition. Help the emperor buy his bread and olives. A Roman numeral can be translated to an Arabic numeral by adding the value of the Roman numeral's letters together. The values of the letters are shown in the table to the right.

I	1
IV	4
V	5
IX	9
X	10
XL	40
L	50
XC	90
C	100
CD	400
D	500
CM	900
M	1000

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set is composed of two Roman numeral fractions separated by a space, the operation to be performed (either addition or subtraction), and another space. The input fractions will always be either a proper fraction or an improper fraction, will always be positive, and will not always be simplified.

Output

For each data set, print out the input expression followed by a space, an equals sign, another space, and the result of the operation as a simplified Roman numeral fraction. The value of the result will never be 0. The resulting fraction should be proper if the absolute value of the result is less than one and improper if the absolute value of the result is greater than one. If the value of the result is an integer, print the result over one; for example, if the value of the result equals one, print `"I/I"`. The range of the absolute value of both the numerator and the denominator of any input and output fraction will be 1 to 3999, inclusive.

Example Input File

```
4
I/I - I/II
III/V + IV/V
XXXII/II - XVI/IV
CCCXXXIII/CVI - XXII/VII
```

Example Output to Screen

```
I/I - I/II = I/II
III/V + IV/V = VII/V
XXXII/II - XVI/IV = XII/I
CCCXXXIII/CVI - XXII/VII = -I/DCCXLII
```

12. Store Window

Program Name: Window.java **Input File: window.dat**

Jill is an employee at a local department store. Her job is to maintain the store window which displays the clothing to entice customers to enter the store, while maintaining the store's budget (which starts at $0). Jill orders the clothing to be displayed at the window directly from the warehouse. She likes to have the most expensive clothing available that she can afford on display. The warehouse does not have an unlimited supply of clothing, so Jill often has to wait for the warehouse to get the next shipment of best clothing. In addition to the warehouse delay, Jill is on a limited budget, and funds for her window clothing will periodically be added to the budget due to customer purchases. In other words, as clothing becomes available, Jill will buy the most expensive piece of clothing she is able to buy if its value is greater than what she currently has at the window. If there is nothing in the warehouse stock that is more expensive than what is at the window, Jill will keep the current item at the window until either a better piece of clothing becomes available or Jill has gathered enough funds to purchase more valuable clothing for the window.

Jill has to keep a log throughout the year of what clothing she puts at the window to submit to her managers. The log simply consists of a record of every change in the clothing at the window that occurs over a 365 day period.

Input

- The first line will contain a single integer `n` that indicates the number of data sets that follow.
- Each data set will begin with two integers `b` and `c`. `b` is the number of increases of funds Jill receives in that year, and `c` is the number of items of clothing that will become available at the warehouse that Jill can buy from.
  - The next `b` lines will consist of the increases in funds that Jill will receive. Each increase in funds is in the format `"day amount"` where:
    - The integer `day` is the day that the funds become available, where $1 \leq$ `day` ≤ 365.
    - The integer `amount` is the amount of money that is to be added to Jill's clothing budget, where $1 \leq$ `amount` ≤ 10000.
  - The next `c` lines will consist of the clothing that will become available from the warehouse. The clothing entries are in the format `"day price name"` where:
    - The integer `day` is the day the clothing becomes available for Jill to purchase from the warehouse, where $1 \leq$ `day` ≤ 365.
    - The integer `price` is the amount of money Jill must pay for the item of clothing, where $1 \leq$ `price` ≤ 10000.
    - The string `name` is a word or phrase which defines the clothing.

Output

For each data set, output Jill's log for the window. There is a new log entry for each time Jill changes the clothing at the window. Each log entry includes the day of the year that the change occurred and the name of the clothing that was displayed at the window. Each log entry is on its own line. The last line should be the amount of money in the budget left in the format `"The budget has `*`v`*` left."` where `v` is the amount of money left over at the end of the year. A blank line should separate each data set.

Example Input File

```
2
6 5
1 30
73 50
146 30
219 20
292 60
300 50
50 10 Ugly Cap
55 60 Cool Cap
100 80 Creepy Trench Coat
200 1 pre-torn pants
330 100 fancy pants
8 7
59 28
137 90
168 87
186 71
200 65
211 20
335 43
365 11
55 36 Luigi Cap
76 195 Bunny Hood
99 181 Gravity Suit
108 33 Green Tunic
133 1 Arm Cannon
180 71 Mario Cap
344 97 Overalls
```

Example Output to Screen

```
Day 50 Ugly Cap
Day 73 Cool Cap
Day 292 Creepy Trench Coat
The budget has 90 left.

Day 133 Arm Cannon
Day 137 Luigi Cap
Day 180 Mario Cap
Day 200 Bunny Hood
The budget has 112 left.
```