University Interscholastic League

Computer Science Competition

2008 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 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	Bingo or Bongo?
Problem 2	Block Down
Problem 3	ebay®
Problem 4	Goldbach's Conjecture
Problem 5	Hangman
Problem 6	Largest Square
Problem 7	Minutes Played
Problem 8	Musical Chairs
Problem 9	Savings
Problem 10	UIL
Problem 11	Unlucky in Love
Problem 12	Valet Parking

1. Bingo or Bongo?

Program Name: Bongo.java Input File: bongo.dat

Bingo is a game played with 24 integers in the range [1, 75] placed randomly in a 5 x 5 matrix on a card and a free space placed in the middle square of the card. The five columns are named B, I, N, G, and O respectively from left to right. For a card to be valid, it must meet the following requirements:

B column can only have integers 1 through 15
I column can only have integers 16 through 30
N column can only have integers 31 through 45
G column can only have integers 46 through 60
O column can only have integers 61 through 75
There is exactly one free space, it is in 3rd square of the N column, and is marked by FS on the card.

B	I	N	G	O
3	17	33	49	64
6	21	44	56	73
14	25	FS	59	69
9	16	45	46	61
5	30	37	60	70

When Bingo is played, 75 balls are placed in a ball machine and stirred. The balls are ejected from the ball machine one at a time and the Caller calls the letter and number of the ball ejected in the order that they are ejected. A player marks the numbers called on his card as the numbers are called.

You are to write a program that checks a bingo card after all balls listed have been called to determine if the player had a Bingo or a Bongo (the card was not a Bingo). For a Bingo, the card must have:

- 5 numbers that were called in any row, any column or either diagonal
- no more balls were called after his bingo was complete.
- The free space may be used to complete any row, column or diagonal in which it falls. For example. B3, I21, FS, G46, and O70 would be a Bingo (if the person calls Bingo on the last number that completes his Bingo) because they are all on one diagonal.

Input

The first line will contain a single integer `n` that indicates the number of Bingo games to follow. For each game, the first five lines will contain the bingo card and the 6th line will contain the letter and the number of the balls in the order that they are called. All bingo cards and bingo balls will be valid and no bingo balls will be repeated.

Output

If the card has a Bingo, you will print `BINGO` and if the card does not have a Bingo, you will print `BONGO`.

Example Input File

```
2
1 16 31 46 61
2 17 32 47 62
3 18 FS 48 63
4 19 34 49 64
5 20 35 50 65
B12 I18 G48 O68 O63 B9 B3
1 19 35 47 63
12 21 41 46 73
7 15 FS 59 61
14 29 34 55 70
2 30 40 60 68
I21 B14 B2 B9 I24 G55 G54 O74 O62
```

Example Output to Screen

```
BINGO
BONGO
```

2. Block Down

Program Name: Block.java Input File: block.dat

The game Block Down takes place on a rectangular grid. The grid can be described as follows:

- Each element in the grid is either empty or occupied by a block.
- The blocks are one of five colors. The colors are red, green, blue, yellow, and purple. Colors are represented using the letters R, G, B, Y, and P.
- Empty cells are indicated with a period.
- A grouping is a set of blocks of the same color that are *connected* to one another.
  - A block is connected to the blocks above, below, to the left, and to the right of it.
  - Connections wrap around so it is possible for a block in the far right column to be connected to blocks in the far left column and for blocks on the bottom row to be connected to blocks on the top row.
  - Connections cannot pass through empty cells.

Here is an example board with five rows and four columns per row. In this example below the second row is `RGGR`. The `R` in the first column is connected to the `R` in the fourth column and vice versa.

```
.RGG
RGGR
BPGY
GBRB
PBBP
```

The Block Down game plays by the following rules:

- A players starts with 0 points.
- A player plays the game by selecting a position in the grid indicated by a row and column, called a block.
- If a player selects an empty block nothing happens and their score is not altered.
- If the block a players selects is part of a grouping of two or fewer blocks they lose one point.
- If the block is part of a grouping of three or more blocks:
  - The player is awarded points equal to the number of blocks in the grouping squared.
  - All of the blocks in the grouping are removed.
  - Any blocks above the removed cells move down towards the bottom of the grid.

For example, suppose the player selects the block in the fourth row and second column of the example board. This block is colored blue. That block is part of a grouping of three blue blocks as shown below, where the grouping is in bold.

```
.RGG
RGGR
BPGY
G**B**RB
P**BB**P
```

For this selection, the player would get nine points, the three blue blocks would be removed and the grid would now look like this.

```
...G
R.GR
BRGY
GGGB
PPRP
```

You are to write a program to simulate the game of Block Down based on an initial set up and a series of player moves.

Input
The first line of input will contain a single integer `n` indicating the number of data sets to process. The remainder of the input consists of those `n` data sets.

Each data set will consist of four parts. The first part will be a line with two integers `r` and `c`. The integer `r` indicates the number of rows and `c` indicates the number of columns per row in the grid for this data set. The next `r` lines are the initial setup of the grid. Each row will have `c` characters. All of the characters will be either a period or R, G, B, Y, or P. The initial setup will be correct in that there will not be any empty cells with blocks above them.

The next line of the input will contain a single integer `m` indicating how many cells the player picks in this data set before giving up. Note that a player may keep making picks even if there aren't moves left that can give them points. The next `m` lines are the player's move for this data set. Each line contains two integers `row` and `col` indicating the row and column of the players next pick. The player numbers the top row of the board 1 and the left most column 1. Moves outside the bounds of the board will not occur in the input.

Output
For each data set output the player's final score based on the initial board, an initial score of zero, and the picks the player makes. It is possible to have a negative score. Print the score on a line by itself. After the score print the final state of the board for that data set.

Example Input File
```
2
5 4
.RGG
RGGR
BPGY
GBRB
PBBP
4
1 1
1 2
4 2
5 1
2 7
RGGGGGR
RRRBRRR
6
2 4
1 2
1 1
2 4
1 1
2 4
```

Example Output To Screen
```
17
....
..GG
R.GR
BRGY
GGRB
86
.......
...B...
```

3. ebay®

Program Name: Ebay.java Input File: ebay.dat

The online auction company, ebay, has just announced that they have reduced their insertion fees. An insertion fee is the price ebay charges the seller for placing an item for sale on ebay and is the minimum price that the seller will accept for the listed item. However, to compensate for that increase, ebay has raised their final value fees. A final value fee is the fee that ebay charges the seller for items sold on ebay. The final value fee is based on the closing value of the sale (the price the buyer paid the seller for the item).

Below is the insertion fee chart that ebay used to charge and what they charge now:

Starting Price	Old Insertion Fee	New Insertion Fee
$0.01 - $0.99	$0.20	$0.15
$1.00 - $9.99	$0.40	$0.35
$10.00 - $24.99	$0.60	$0.55
$25.00 - $49.99	$1.20	$1.00
$50.00 - $199.99	$2.40	$2.00
$200.00 - $499.99	$3.60	$3.00
$500.00 or more	$4.80	$4.00

Below is the final value fee chart that ebay used to charge and what they charge now:

Closing Value (CV)	Old Final Value Fee	New Final Value Fee
Item not sold	No fee	No fee
$0.01 - $25.00	5.25% of the CV	8.75% of the CV
$25.01 - $1000.00	5.25% of the initial $25 plus 3.25% of the CV from $25 to $1000	8.75% of the initial $25 plus 3.5% of the CV from $25 to $1000
$1000.01 and up	5.25% of the initial $25 plus 3.25% of the CV from $25 to $1000 plus 1.5% of the CV $1000.01 and up	8.75% of the initial $25 plus 3.5% of the CV from $25 to $1000 plus 1.5% of the CV $1000.01 and up

You are to write a program that will, given the starting price and closing price, compare the total fee (insertion fee + final value fee) that would have been charged the seller before and after ebay's price reduction.

Input

The first line will contain a single integer `n` that indicates the number of items the seller listed on ebay. Each of the next `n` lines will contain the starting price of an item, a space, and either the sale price (closing value) or the words `NOT SOLD` if no one bought the item.

Output

For each item, you will print whether the new total value fee is `MORE`, `LESS`, or the `SAME` as the old total value fee would have been, followed by the amount of that difference rounded to the nearest penny in the format below. Round only the difference between the old total value and the new total value; do not round any intermediate values.

Example Input File

```
3
49.99 75.10
9.99 999.99
75.00 NOT SOLD
```

Example Output To Screen

```
More $0.80
More $3.26
Less $0.40
```

4. Goldbach's Conjecture

Program Name: Goldbach.java Input File: goldbach.dat

Goldbach's conjecture can be expressed as follows: Every even integer greater than or equal to 4 can be expressed as the sum of two prime integers. This conjecture has neither been proved nor disproved. [Note: An integer is considered prime if it is divisible only by 1 and the integer itself. The number 1 is not considered prime in this definition.]

In this problem you will be presented with several even integers greater than 4. For each even integer you will write the unique pairs of prime integers that add up to that even integer. The pairs have to be unique, i.e. if you find a pair (p1, p2) then you cannot claim that (p2, p1) is another pair that add up to that even integer.

Input

The first line will contain a single integer `n` that indicates the number of even integers greater than 4 that follow. The next `n` lines will each contain a single even integer greater than 4. There will be no even integer greater than 100.

Output

For each even integer you will write out all the unique pairs of prime numbers that add up to that number.

Example Input File

```
3
8
20
42
```

Example Output to Screen

```
8 = 3 + 5
20 = 3 + 17 = 7 + 13
42 = 5 + 37 = 11 + 31 = 13 + 29 = 19 + 23
```

5. Hangman

Program Name: Hangman.java Input File: hangman.dat

You are going to write a hangman game that plays by the following rules:

- The contestant tries to determine the solution to a puzzle by guessing one letter at time.
- The contestant continues guessing letters until either all of the letters in the puzzle have been guessed, more than seven incorrect guesses have been made or the contestant quits.
- If the letter guessed, whether uppercase or lowercase, is found in the puzzle, each instance of that letter that appears as either an uppercase or a lowercase letter is considered to have been found.
- If the contestant guesses all of the letters before reaching 7 incorrect guesses, the contestant wins.
- If the letter guessed is not in the puzzle, the number of incorrect guesses left will be decreased by one.
- If the contestant has 7 incorrect guesses before guessing all of the letters, the contestant loses.
- If the contestant runs out of guesses before winning or losing, the contestant is considered to have quit.
- You may assume the contestant will not guess the same letter more than once.

Input

The first line will contain a single integer `n` that indicates the number of games that will follow. For each game, the first line will contain a word or phrase that is the puzzle to be solved. The second line of the game will contain the contestant's guesses. Only alphabetic characters and spaces will appear in a puzzle.

Output

For each game, output `WON` if the contestant won, `LOST` if the contestant lost, or `QUIT` if the contestant quit before winning or losing followed by the number of incorrect guesses the contestant had left.

Example Input File

```
3
Java is just fun
aeistfunjv
Lickety split
aeioubcdfg
UIL Academics
AIOUqwerty
```

Example Output to Screen

```
WON 6
LOST 0
QUIT 1
```

6. Largest Square

Program Name: LSquare.java Input File: lsquare.dat

Given a square grid of uppercase letters, you are to find the largest square formed around a given location called the center. The location of each character in the grid is given by an ordered pair (r,c) that indicates the row r and the column c of a character relative to the origin. The origin is the top left position of the grid and is considered to be position (0,0). A Largest Square is the largest contiguous s x s area that contains only the same letter as its center.

Input

The first line will contain a single integer n that indicates the number of data sets to follow.

The first line of each data set will contain a single integer p that indicates the number of rows and columns in the square matrix of uppercase letters to follow. The next p lines will contain the p x p square matrix. The following line will contain a single integer m that indicates the number of center locations (r, c) to follow. Each of the next m lines will contain two integers r and c that represent the center for which you are to find the largest square.

Output

For each center of each data set, you are to print r c s where r and c are the row and column of the center of the Largest Square and s is the number of letters in an edge of the Largest Square that contains only the same letter as its center. At least one blank line should be printed between sets of output.

Example Input File

```
2
9
AAABBBAAA
AAABBBAAA
AAABBBAAA
AAABBBAAA
AABBBBBBB
AABBBBBBB
ABBBBBBBB
BBBBBBBBB
BBBBBBBBB
3
3 2
1 1
6 5
5
BBBBB
BBBBB
BBBBB
BBBBB
BBBBB
2
1 2
2 2
```

Example Output To Screen

```
3 2  1
1 1  3
6 5  5

1 2  3
2 2  5
```

7. Minutes Played

Program Name: Minutes.java Input File: minutes.dat

One of Jennifer's jobs as an assistant to the girls basketball coach is to keep track of the number of minutes that each girl on the team plays. There are four quarters in a game and the time clock starts at 12:00 (12 minutes, 0 seconds) at the beginning of each quarter and counts down to 0:00 (0 minutes, 0 seconds). Jennifer records the minutes and seconds left in the quarter each time a player enters or leaves the game. You are to write a program that calculates the total playing time of each player.

Input

The first line will contain a single integer `n` that indicates the number of lines to follow. Each of the next `n` lines will contain the first name of a player followed by one of more data sets in the form `Qx mm:ss Qy mm:ss`. The quarter that the player entered the game is denoted by `x` and is followed by the number of minutes and seconds left in that quarter when she entered the game. The quarter the player exited the game is denoted by `y` and is followed by the number of minutes and seconds left in that quarter when she exited the game.

Output

For each player, print the player's name followed by the number of minutes and seconds that the player played during the game in the format shown in the **Example Output** below.

Assumptions for both Input and Output

The number of minutes will be in the range [0,12] and have no leading zeroes. The number of seconds will always be a two digit integer in the range [00,59].

Example Input File

```
4
Mary Q1 12:00 Q1 2:13 Q2 10:45 Q2 7:33 Q2 4:12 Q3 4:33 Q4 9:27 Q4 0:00
Anne Q1 5:55 Q2 7:34 Q3 5:58 Q4 7:55
Suan Q1 9:21 Q2 10:12 Q2 2:21 Q2 1:08 Q3 12:00 Q3 4:03 Q4 3:13 Q4 0:00
Evie Q2 1:10 Q3 9:21
```

Example Output To Screen

```
Mary 34:05
Anne 20:24
Suan 23:32
Evie 3:49
```

8. Musical Chairs

Program Name: Chairs.java Input File: chairs.dat

In the childhood game of Musical Chairs, chairs are placed in a circle with one less chair than players. Children march to the music around the circle of chairs until the music stops. Then each child sits in the chair nearest them. The one child that cannot find a chair is eliminated from the game. A chair is removed and the game continues until there is just one person left. That person is the winner.

You are to write a program to simulate a game of Musical Chairs and determine the winner of each game played. The simulation will work as follows:

- The chairs are numbered from `1` to `m`.
- The children are numbered from `1` to `m+1`.
- You are given two integers `a` and `b`. The first integer `a` is the chair number where child #1 will sit when the music stops. When the music stops, the children begin sitting down with child #1 sitting in chair `a`, child #2 sitting in chair `a+1`, etc, until chair `b` is reached. The person who would have been assigned to chair `b` is eliminated and the next person in line is seated in chair `b`.
- Since the chairs are in a circle, once chair `m` is reached, the children begin sitting in chair `#1` and continue until the rest of the children have been seated.
- The children are then renumbered with the child sitting in chair #1 becoming child #1, the child in chair #2 becoming child #2, etc.
- Chair number `m` is then removed and the game is continued with one less chair and one less child.

Input

The first line will contain a single integer `n` that indicates the number of games to be played. For each game there will be `m+2` lines where `m` is the number of chairs in the game. The first of these lines will be the integer `m` that indicates the number of chairs in the game. The second of these lines will have the first names of `m+1` players separated by a comma and a space. Each of the next `m` lines will have two integers separated by a space. The first integer `a` will represent the chair number where the children will begin sitting and the second number `b` will be the number, relative to the first chair, of the person who will not find a seat.

Output

For each game, you will output the order of the students, beginning with chair #1, each time the music stops. The winner will be the last child. Place whitespace between games.

Example Input File

```
1
6
ANNE, BECKA, CHAD, DREW, ERNIE, FRANK, GEORGE
3 5
2 1
4 3
3 1
2 2
1 2
```

Example Output To Screen

```
FRANK GEORGE ANNE BECKA DREW ERNIE
ERNIE FRANK GEORGE ANNE BECKA
FRANK GEORGE BECKA ERNIE
BECKA ERNIE FRANK
FRANK ERNIE
FRANK
```

9. Savings

Program Name: Savings.java **Input File: savings.dat**

South Star Bank has a debit card program that automatically deposits money from your checking account into your savings account. Each time you use your debit card, if the amount of your purchase is not a whole dollar amount, the bank charges your checking account the next highest dollar amount over the amount of your purchase and deposits the difference between the amount of your purchase and the next highest dollar amount to your saving account.

For example, your purchase is for $8.15. The next highest dollar amount is $9.00 and that amount will be deducted from your checking account. The difference between $8.15 and $9.00 is 85 cents so 85 cents will be deposited into your savings account. If your purchase is $12.00, that is a whole dollar amount and no money would be deposited into your savings.

You are to determine the total amount of money that will be deposited into your savings account after a series of purchases.

Input

The input file contains an unknown number of lines. Each line in the input file will have the amount of a single purchase.

Output

Print the total amount of money that will be placed in your savings account. There must be a leading dollar sign and the amount rounded to the nearest penny. You may assume the amount will be more than one dollar.

Example Input File

```
12.34
34.78
8.01
```

Example Output to Screen

```
$1.87
```

10. UIL

Program Name: UIL.java Input File: none

Nicholas wants to print the letters UIL to use as a pattern for an art project. You are to print his design as shown below.

Input
There is no input for this problem.

Output
You are to output the letters `UIL` with 20 rows as shown below.

Output to Screen
```
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UU     UU    II    LL
UUUUUUUU    II    LLLLLLLL
UUUUUUUU    II    LLLLLLLL
```

11. Unlucky in Love

Program Name: Unlucky.java Input File: unlucky.dat

A number is defined to be *unlucky* if its leading and trailing sums both equal 13. A leading sum is obtained by adding together the first `n` decimal digits of the number. A trailing sum is obtained by adding together the last `m` decimal digits of a number. Different leading and trailing sums can be formed from a single number given different values of `n` and `m`. A number is considered unlucky if there is a value for `n` such that the leading sum of those `n` digits is 13 and there is a value for `m` such that the trailing sum of those `m` digits is also 13.

Input
The first line of input will contain a single integer `n` indicating the number of data sets to process. The following `n` lines will consist of decimal numbers, one number per line.

Output
For each number in the input print `UNLUCKY` if the number is an unlucky number or `NOT UNLUCKY` if it is not an unlucky number.

Example Input File
```
5
13
2173555555552902
6767
30034120006
123456789101112
```

Example Output To Screen
```
NOT UNLUCKY
UNLUCKY
UNLUCKY
UNLUCKY
NOT UNLUCKY
```

12. Valet Parking

Program Name: Valet.java Input File: valet.dat

Drew is working his way through college as a parking lot valet. The "lot" he uses to park the cars is really a long, narrow alleyway. Therefore, to retrieve a car in the middle, he must first move the cars blocking it. Since he can get a car from either end of the alleyway, he always chooses the end that has the fewest cars that he will have to move to get to the car he needs. If there is an odd number of cars parked in the alleyway and he needs the middle car, he always chooses to get the car from the front of the alleyway. The process for his getting a car is to determine if he will get the car from the front or rear of the alleyway, then move the cars that are blocking the needed car, move the needed car, and then return the moved cars to the alleyway in the *reverse* order.

For example, if the cars ABCDEFG are in the alleyway with A being in front and G being in the rear, and he needs to retrieve car E, he must first remove car G and then remove car F before retrieving car E. Then he must return the cars back in reverse order so he first returns G and finally returns car F. So, the order of the remaining cars would be ABCDGF.

You are to write a program that will give John the order of the cars at any given time.

Input
The first line contains a single integer `n` that indicates the number of cases. Each of the next `n` lines will contain a string of distinct letters indicating the original parking configuration, followed by a list of single letters denoting the order that the cars are being picked up. There will always be at least one car to be picked up.

Output
Print the original parking order on one line. Then, on separate lines, print the letter of the car being picked up and the order of the cars remaining. If all cars have been picked up, print `PARKING LOT EMPTY`. Separate each case with whitespace.

Example Input File
```
2
ABCDEFG E C D
REGION N O I E R G
```

Example Output to Screen
```
ABCDEFG
E ABCDGF
C BADGF
D ABGF

REGION
N REGIO
O REGI
I REG
E RG
R G
G PARKING LOT EMPTY
```