University Interscholastic League

Computer Science Competition

2013 Regional Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

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	Best Day of Sales
Problem 2	Bit Packing
Problem 3	Bouncing Dice
Problem 4	DQL
Problem 5	Florist Shop Displays
Problem 6	Golf Qualifying
Problem 7	Maze 3D
Problem 8	Moose
Problem 9	Multiple Choice
Problem 10	Ping Pong Tournament
Problem 11	Thirteen Jewels
Problem 12	Waitlist

1. Best Day of Sales

Program Name: BestDay.java   Input File: bestday.dat

James owns a music store that he keeps open seven days a week. He would like to close one day a week to spend more time with his family. In an effort to determine which day would be the best for him to close his business, he has collected the statistics on his sales for each day for the past few weeks.

You are to write a program that will find which day of each week that his sales are the highest.

Input

The first line of input will contain a single integer `n` that indicates the number of weeks of data. Each of the following `n` lines will contain 7 integer values, each separated by a space, representing James' sales on days `SUNDAY` through `SATURDAY` respectively.

Note: You may assume that no two days will have the same amount of sales.

Output

For each week and in uppercase, you will print the day of the week that his sales were the greatest, as shown below.

Example Input File

```
4
1300 1500 1200 1600 1800 900 1400
1200 1400 1500 1600 1100 1450 1475
1745 2534 2000 2100 2400 1975 1823
2231 1992 2000 2345 2435 1982 2500
```

Example Output to Screen

```
THURSDAY
WEDNESDAY
MONDAY
SATURDAY
```

2. Bit Packing

Program Name: BitPacking.java   Input File: bitpacking.dat

A small specialty store in town wants you to write a program that can read in data from their specialized inventory database and print out the information in human readable format so they know what they have in stock and what they need to order. Because they are charged by the size of their database, they have condensed their data into single 32 bit integers whose bits contain 3 unique pieces of information: the item's type, the item's sub-type, and the amount of the item left in stock. The data is stored as follows:

- The top 9 bits are currently unused, and can be assumed to be `0`.
- The next 5 bits contain the inventory item's type, will be a value from `1` to `26`, and will correspond to the uppercase letters of the alphabet, `A` through `Z`, in order.
- The following 6 bits contain the inventory item's sub-type and will range in value from `1` to `61`, where `1` to `26` are the uppercase letters `A` through `Z`, `27` through `52` are the lowercase letters `a` through `z`, and the values `53` through `61` are the numbers `1` through `9`.
- The final bits represent the amount of the item they have in inventory.

Input

The first line of input will contain a single integer `n` that indicates the number of packed bit values that will follow. Each of the following `n` lines will contain a single integer (in base 10) representing a packed bit value.

Take, for example the first input value `536839`, converted to binary and cut up into 4 groups using the problem description:

```
    000000000 00010 000011 000100000111
        0       2      3         263
```

The first group of nine bits converts to decimal 0 and is unused. The second group of 5 bits is type 2, corresponding to `B`; the third group of 6 bits is subtype 3, corresponding to `C`; and the fourth and final group of bits is `263`, the amount of inventory they have.

Output

For each packed bit value you will print out a single line in the form

```
Type: X | Sub Type: Y | Inventory: Z
```

Where *X* is the upper case character denoting the type, *Y* is the upper case, lower case or numerical value of the sub-type, and *Z* is the amount they have in inventory, in base 10.

Example Input File

```
2
536839
6672394
```

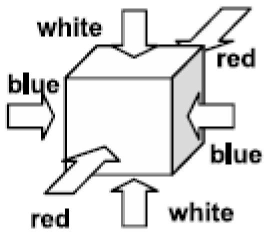
Example Output to Screen

```
Type: B | Sub Type: C | Inventory: 263
Type: Y | Sub Type: c | Inventory: 10
```

3. Bouncing Dice

Program Name: Bouncing.java Input File: bouncing.dat

You are given a single six-sided die as shown on the right. Two of the opposite sides are colored white, two are colored blue, and two are colored red as shown in the picture to the right.



You are to write a simulation that will determine the color that is facing upward after a given number of "rolls". At the beginning of each simulation, the die will be positioned as shown at the right:

- White sides are on the top and bottom.
- Blue sides are on the left and right.
- Red sides are on the front and back.

For each simulation, you will need to construct one object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

For each simulation, you will need three random numbers: `r1`, `r2` and `r3`:

- The first random number, `r1`, will be a number between 1 and 10, inclusive, and will determine the number of times the direction will change once the die is rolled.
- For each of `r1` actions, there will be two random numbers created, `r2` and `r3`.
- `r2` will be a number between 1 and 4, inclusive, and will deterrmine the direction that the die will roll:
  - 1 = forward (red would be up on the first roll)
  - 2 = backward (red would be up on the first roll)
  - 3 = left (blue would be up on the first roll)
  - 4 = right (blue would be up on the first roll)
- `r3` will be a number between 1 and 5, inclusive, and will determine the number of times the die will roll in the direction determined by `r2`.

The simulation will take the original die, roll it in direction `r2` `r3` number of times then, from that result, repeat the process `r1` number of times.

Input

The first line of input will contain a single integer `n` that indicates the number of simulations to follow. Each of the following `n` lines will contain a positive long which will be the seed for the random object in the simulation.

Output

For each simulation, you will print the color that will be on the top side of the die at the conclusion of the simulation. The color that will be printed will be `RED`, `WHITE`, or `BLUE`.

Example Input File

```
2
12354643223452
76654213645
```

Example Output to Screen

```
WHITE
RED
```

Random numbers generated: For each seed, the first number is `r1`. The remaining numbers are alternating `r2` and `r3`.

```
12354643223452: 10  3 1 4 3 3 5 4 2 4 1 1 5 2 5 3 1 1 2 3 5
76654213645: 6  2 4 1 4 4 5 3 4 2 5 4 2
```

4. DQL

Program Name: DQL.java Input File: dql.dat

Your boss needs you to write a parser for the Data Query Language specification for your company's database software. Your program will also test your parser by reading in a sample database, running some queries and printing the results of those queries to the screen. A database is a table, where each column has a header label, and each row is a database entry consisting of the values for each column.

The language itself is pretty straight forward, where the query form and query return values are described below:

- Query form: `SELECT <ITEM> WHERE <HEADER><OP><VALUE>`
  - `SELECT` always begins a query
  - `<ITEM>` denotes which columns' data entries to return in the query, and can be:
    - an asterisk (`*`) which means to report the data entries from all columns, or
    - a comma separated list of column names (with a space after each comma) from which you are to report the data entries.
    - If there is no comma, you will report only the data entries from the `<ITEM>` columns meeting the query criteria.
  - `WHERE` starts an optional segment of the query consisting of `<HEADER><OP><VALUE>` which are used to run a logical check against the data entry to see if it should be included in the results:
    - `<HEADER>` is the header label of one of the columns.
    - `<OP>` is a logical operator, either =, <, >, <= or >=. A column's data should be treated as a string if the = operator is used, and an integer otherwise.
    - `<VALUE>` is the value to compare against. It should be treated as a string if the = operator is used, and an integer otherwise.
- Query return value:
  - if `<ITEM>` is an asterisk (`*`):
    - it will return all the data entries in the rows that match the `WHERE` segment,
    - or all the data entries in all rows if the `WHERE` segment is missing.
  - otherwise:
    - it will return the data entries of the columns from the `<ITEM>` part of the query if there is matching data from the `WHERE` segment,
    - or `NONE` if there is no matching data.

Input

The input file contains first the database data and then the queries.

- Database data:
  - The first line of the input file will contain two integers `c r` separated by a space, where the `c` is the number of columns and `r` is the number of database entries.
  - The second line will contain a space delimited list of header labels for each column `c` of the database.
  - The next `r` lines will each contain a space delimited list of values for each column `c` of the database.
- Queries:
  - Following the database data, there will be a line containing a single integer n that indicates the number of queries to follow.
  - Each of the following `n` lines will contain a single query in the format described above.

Output

For each query you will print the string `Query #`*`n`* where *`n`* is the number of the query starting at 1. Then you will print the data from the query, one data entry per row and in the order they appear in the database. If the query returns no data, then you should print `NONE`. There should be a blank line between sets of query output.

(Continued on next page)

4. DQL (cont.)

Example Input File
```
4 5
Name Sex Age Grade
Bob M 18 91
Jan F 17 79
Gus M 18 80
Fern F 18 99
Kat F 17 70
2
SELECT Name, Age WHERE Sex=F
SELECT Name WHERE Grade>90
```

Example Output to Screen
```
Query #1
Jan 17
Fern 18
Kat 17

Query #2
Bob
Fern
```

Note: A blank line at the end of the output is optional.

5. Florist Shop Displays

Program Name: Florist.java   Input File: florist.dat

Susan is the manager of a florist shop. She likes to display her flowers in pots throughout her store. To make the displays more esthetically pleasing, she places the pots on plant stands, one pot per stand, using plant stands of different heights so some displays are taller than others. Unfortunately even though all of the display stands are the same diameter, they have different limits on how much weight they can hold.

You are to write a program that will determine the maximum number of plants Susan can display given the weights of the plants and the weight limits of the plant stands.

Input

The first line will contain an unknown number of space delimited positive integers that represent the weights of the plant stands. The second line of input will contain a single integer `n` that indicates the number of displays to be considered. Each of the following `n` lines will contain a space delimited list of an unknown number of positive integers that indicate the weights of the plants for the display.

Output

In the order of input, you will print the number of plants that can be placed on stands for each display.

Example Input File

```
12 14 5 6 7 12 14 3 12 8
3
5 7 9 12 3 2 12 14 7 8 10
12 4 6 7 15 13 13 15 12 10 7 5 8 7
5 7 10 7 16 7 8 14 15 9 17
```

Example Output to Screen

```
10
9
8
```

6. Golf Qualifying

Program Name: GolfQ.java   Input File: golfq.dat

Your high school golf team is planning to host a golf tournament. They have asked you to write a program that will determine the players that will get to play in the final round of the tournament.

The players that advance to the final round are all the players that have a three day total score lower than the player in 12th place plus anyone who is tied with the 12th place score.

Input

The first line of input will contain a single integer `n` >15 that indicates the number of players who have finished the first three rounds. Each of the next `n` lines will contain a player's name in the form: `l, f s` where `l` is the player's last name, `f` is his first initial and `s` is his three digit, three day total score.

Output

In the order they appear in the original list, you will print the names of the players qualified for the final round of the tournament.

Example Input File

```
15
JONES, J 210
SMITH, S 212
GEORGE, S 220
LILLARD, E 218
GREGORY, E 213
ROGERS, R 215
RICHARDS, O 222
LASITER, B 212
CHRISTIAN, N 219
LOWERY, A 221
RANDALL, R 227
THOMAS, T 221
TRAVIS, R 220
LESTER, P 220
BRIDGES, M 218
```

Example Output to Screen

```
JONES, J
SMITH, S
GEORGE, S
LILLARD, E
GREGORY, E
ROGERS, R
LASITER, B
CHRISTIAN, N
LOWERY, A
THOMAS, T
TRAVIS, R
LESTER, P
BRIDGES, M
```

7. Maze 3D

Program Name: Maze3D.java   Input File: maze3d.dat

NASA is testing a new drone to navigate through densely packed asteroid fields. Because of your world renowned programming skills, NASA has chosen you to write an algorithm that will find the optimal path through any asteroid field so they can make sure their drone is taking the shortest path, thus saving fuel and money. All the simulations testing your algorithm will take place in a 3-dimensional asteroid field which will contain one start point, one end point, empty spaces, asteroids and small debris. The new drone is unique in that it has a 3 shot laser cannon that can blast small debris out of its way.

The following characters will be used to describe the asteroid field for the simulations:

- `.` - Denotes empty space
- `#` - Denotes an impassible asteroid
- `*` - Denotes small debris that can be blasted away with the laser
- `S` - Denotes the start position of the drone
- `E` - Denotes the exit cell

The goal of your algorithm is to find the shortest distance through the asteroid field from the Start `S` to the Exit `E`. The drone cannot move into cells with asteroids, but it can move into cells with debris as long as it has enough laser shots left to clear the debris first. Remember, the drone only has 3 shots, so use them wisely! The drone can only move up, down, left, right, forward and back; it cannot move diagonally in any direction.

Input

The first line of input will contain a single integer `n` that indicates the number of simulations to follow. For each simulation:

- The first line contains three positive integers in the form `r c f`, where `r` is the number of rows, `c` is the number of columns, and `f` is the number of vertical "floors", or height, of the 3-dimensional area.
- Then, for each floor `f`, there will be `r` lines of input, with `c` characters in each, denoting what populates each cell of the three dimensional grid as described above.
- There will be no spaces on any of the lines, nor will there be an empty line between floors.

Output

For each simulation you will output `# MOVES` where `#` denotes the number of moves along the shortest path between the `S` and `E`, or `STUCK` if the drone cannot get to `E`.

Example Input File

```
1
4 4 4
S..#
#..#
#..#
####
.###
...#
...#
####
.###
..##
#..#
#..#
*.*.
###.
###.
###E
```

Example Output to Screen

```
9 MOVES
```

8. Moose

Program Name: Moose.java   Input File: moose.dat

Hermit Joe lives in a desolate area of Alaska. The Alaskan Wildlife Management team has given him several maps of the area surrounding his cabin that show the location of the homes of different types of animals. The map is a 10x10 rectangular grid of the area and the characters on the map represent different animals that live in the area. For example, the letter E represents an eagle's nest, the letter B represents a brown bear den, the letter M represents a moose habitat, etc. Being an avid naturalist, Joe is especially interested in filming moose in their natural habitat for a documentary he is making. He needs you to write a program for him that will find the largest moose habitat on a given map. The largest moose habitat is the area of the map with the most horizontally and vertically contiguous moose characters. The size of the moose habitat is the number of moose characters that form the moose habitat.

Input

The first line of input will contain a single integer n that indicates the number of maps to be checked. Each map will contain ten lines (rows) with ten characters (columns) and no spaces on each row. All characters in the map will be either an uppercase letter of the alphabet that represents an animal's home, or a period (.) that represents an area without a major animal home.

Output

For each map, you will print the size of the largest moose habitat.

Example Input File

```
1
M...E..MMM
...B....MM
EMMCC.BBMM
EB.MMFFBMM
MMMM..B..M
MMM.B..FMM
MMMMFFFMMM
CC.BM..MMM
BECMM...FJ
..CMMFF.B.
```

Example Output to Screen

```
18
```

9. Multiple Choice

Program Name: MultChoice.java   Input File: multchoice.dat

The UIL test writer needs some help with her multiple choice questions that have a group of options followed by five answers with different combinations of the options. For example, consider the question:

Assume that `Tractor` and `Car` are subclasses of the class `Vehicle`, and that all three classes have a no-argument constructor. What choices for **<*1>** will allow the following statement to compile?

```
Vehicle x = <*1>();
```

I. `new Vehicle`	III. `new Car`	V. `new Tractor`
II. `Vehicle`	IV. `Car`	VI. `Tractor`

A. I

B. II, IV, and VI

C. III

D. I, III, and V

E. III and V

The author of the question intends for answer D to be the correct answer since it is the best answer (and the directions say to choose the BEST answer) but some contestants argue that answers A and C are correct as well. To clarify her questions, you have been asked to write a program that will add the word "only" to all answers that need to be clarified.

You have determined that answer choices A, C, D, and E all need the word "only" added to end of the choices because answer A is contained in answer D, answer C is contained in answer D, and both choices in answer E are contained in answer D. So the answer choices should read:

A. I only

B. II, IV, and VI

C. III only

D. I, III, and V

E. III and V only

Input

The first line of input will contain a single integer `n` that indicates the number of sets of five answers to follow. Each answer will begin with a capital letter `A` through `E` followed by a period, a space and a list of comma and space delimited list of Roman numerals, the last of which will be preceded by the word `and` and a space as shown in the Example Input.

Output

For each set of answers, you will first print `Question` *x*, where *x* is the problem set number. On the next five lines, you will print the answer choices with the word `only` printed at the end of the answers that need clarification as described above. Your output should be formatted as shown in the Output to Screen with a blank line separating each problem set.

(Continued on next page)

9. Multiple Choice (cont.)

Example Input File

```
2
A. I
B. II, IV, and VI
C. III
D. I, III, and V
E. III and V
A. I
B. II
C. III
D. I and II
E. I, II, and III
```

Example Output to Screen

```
Question 1
A. I only
B. II, IV, and VI
C. III only
D. I, III, and V
E. III and V only

Question 2
A. I only
B. II only
C. III only
D. I and II only
E. I, II, and III
```

Note: A blank line at the end of the output is optional.

10. Ping Pong Tournament

Program Name: PingPong.java Input File: pingpong.dat

A seeding method called Swiss Seeding was first used in Zurich, hence called Swiss Seed, to seed a chess tournament in 1895. This seeding method is frequently used in chess, squash, scrabble and other tournaments in which there is not enough time or facilities to play a round-robin type tournament to determine the seeding for the tournament. The Swiss Seeding process allows all players (or teams) to participate in several rounds of competition without being eliminated and without playing any person twice before being seeded for head-to-head single elimination. This method also allows players of similar ability to play each other in the initial rounds.

Your school is hosting a ping pong tournament in the near future and as an incentive for players to attend, you want to guarantee them that they will be able to play at least four games before being eliminated. You are to write a program that will do the Swiss Seeding for your tournament. Your facility will accommodate no more than 20 players.

The process for this Swiss Seeding for the ping pong tournament is:

1) Randomly select the order for the players in round one. This order will establish their relative seed order for the remaining rounds.
2) For round 1, players will play in pairs with player 1 playing player 2, player 3 playing player 4, etc.
3) After a round is finished, players will be reseeded for the next round using the following rules:
   a) The players with the best win-loss record will be listed first in the order of their seeding in round 1 followed by the players with the next best win-loss record in the order of their seeding in round 1, and continuing until all players have been placed in the list.
   b) Players from this list will be paired as described above in item 2.
   c) After the pairings are established and beginning with the first pair, if the first player listed in the pair (I will call him player A) has already played the second player listed in the pair (I will call him player B), you will search in order down the list and:
      (1) find the first player that has not yet played player A (I will call him player X)
      (2) move player X into the list where player B was
      (3) move all players B through player the player ahead of player X down the list while maintaining their order
   d) Repeat c) until all pairs have been checked and processed.
   e) If the last pair of players had previously played each other, player A in this last pairing will switch places with the lowest seed above him that he has not yet played and whose opponent has not played player B.
4) The seedings for round 2, 3 and 4 will be made using steps (a) through (e) above.
5) Round 4 will be the first elimination round.

Input

The first line of input will contain a single integer `n` that indicates the number of test cases that you will have. For each of the test cases:

- The first line will contain a single even integer `m` that indicates the number of players in the tournament.
- Each of the following `m` lines in the test case will contain, in their initial random order:
  - the unique first name of a player with no spaces followed by a space and three characters, each separated by a space.
  - The three characters from above will be either an `L` for lose or `W` for win which will indicate the outcome of their games in each of the three games.

Output

For each test case, you will print `TEST CASE #`*`x`* `ROUND` *`y`*`:`, where *x* is the test case number and *y* is the round number, on a single line and followed by the list of names in their seeding order for rounds 2, 3, and 4 using the format below. Print a blank line after each test case.

10. Ping Pong Tournament (cont.)

Example Input File

```
1
8
Randy W W W
Abel L W L
Max L L L
Robin W L W
Yang W L W
Peggy L L W
Ruby L W L
Andy W W L
```

Example Output to Screen

```
TEST CASE #1 ROUND 2:
Randy
Robin
Yang
Andy
Abel
Max
Peggy
Ruby

TEST CASE #1 ROUND 3:
Randy
Andy
Abel
Robin
Yang
Ruby
Max
Peggy

TEST CASE #1 ROUND 4:
Randy
Yang
Robin
Andy
Abel
Peggy
Ruby
Max
```

Note: A blank line at the end of the output is optional.

11. Thirteen Jewels

Program Name: Thirteen.java Input File: thirteen.dat

In 2013, John's fledgling company created a game app which he named Thirteen. For the game, there will be thirteen jewels placed in a 10x10 grid. The object of the game is for the player to collect all of the jewels as quickly as possible and then exit the grid. A grid will contain the following characters:

- `*` – jewels that the player is to collect.
- `#` – obstacles and the player cannot move into that cell.
- `.` – empty places on the grid to which the player can move.
- `S` – player's start position at time zero. The timer starts when he moves out of this cell.
- `E` – player's exit position. The timer stops when he moves into this cell.

The player will always start in the upper left corner of the grid, collect all of the jewels as quickly as possible, and then exit the lower right corner of the grid. When a player moves to the cell containing a jewel, he collects the jewel and the cell becomes empty.

The game is timed and very fast paced. The solution to the game is the path from the start cell to the exit cell while collecting all thirteen jewels along the way. The player starts in the upper left corner at time zero. He can move horizontally or vertically to any cell that is not an obstacle and that he hasn't visited before, and will exit in the lower right corner once he has collected all of the jewels. You have been assigned to write a program that will find the shortest path solution to the game. Once you find the shortest path, each time you move to a new cell in the shortest path, the timer counts one second. The player will then be given twice as long to try to win the game.

Notes:

- There will always be a path from `S` to `E` that will go through all 13 jewels.
- The data sets have been designed so execution time will be less than 15 seconds – even on a slow machine. Judges should allow 2 minutes.

Input

The first line of input will contain a single integer `n` that indicates the number of games to follow. Each game will contain one grid of 10 rows with 10 characters in each row. The characters in each row will be one of the characters listed above.

Output

For each game, you will print the number of seconds necessary to complete the shortest path.

(Continued on next page)

11. Thirteen Jewels (cont.)

Example Input File

```
2
S.########
..*.....*.
.######..*
####..*..#
##*.....##
..*.######
###.....*.
###*..*##.
##.*..*...
..*..*...E
S...#####.
##*****.#.
######..##
###....###
###..**.##
###.......
#.*..##..*
.#...**.##
#..*....##
##.####.*E
```

Example Output to Screen

```
40
36
```

12. Waitlist

Program Name: Waitlist.java   Input File: waitlist.dat

Your UIL team is hosting an invitational meet to raise money for your team's annual expenses. Owing to room sizes, grading constraints, and other concerns, you are required to limit the number of contestants in each event. Registration is on a first come, first served basis but you are willing to keep a "waitlist" of people to replace contestants who sign up but do not attend. The number of people on the wait list is the number of people wishing to enter the contest less the number of seats allocated for the contest.

You are to write a program that will report the waitlist status of each event of the contest.

Input

The first line of input will contain a single integer `n` that indicates the number of events in the contest. Each of the following `n` lines will contain the name of a contest (with no spaces) followed by a space and two integers `m r` where `m` represents the maximum number of students that can compete in the contest and `r` represents the number of students registered.

Output

In the order of the events listed in the data file, you will print the name of the event followed by a space and the number of students who will be placed on the waitlist or `0` if there is no one on the waitlist.

Example Input File

```
4
Accounting 35 40
ComputerScience 150 110
ReadyWriting 25 28
Spelling 50 57
```

Example Output to Screen

```
Accounting 5
ComputerScience 0
ReadyWriting 3
Spelling 7
```