University Interscholastic League

Computer Science Competition

2004 State 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 6 points.

II. Unless otherwise noted in the problem statement, the following assumptions can be made regarding the input files:

1. Integers will be in the range supported by the native 'int' type.
2. There will be no trailing spaces at the end of any line.
3. For problems with multiple datasets, there will be no blank lines separating datasets.
4. There will be exactly one newline character immediately preceding the end of each input file.
5. Problems will have no more than 50 data sets each.
6. Input files will have no more than 100 lines.

Problem #1: Average Joe

Program Name: average.java Input File: average.dat

At the end of a grading period, computer science teacher Joe Schmoe has to calculate grades for his students. Unfortunately, he does not keep a grade book and only has a big list of student names and test scores. Since he is a computer science teacher, he decided to write a program to calculate student averages and assign letter grades according to the following scale:

93+	A
83-92	B
73-82	C
60-72	D
0-59	F

Input

Since Joe teaches multiple classes, the first line of the input file indicates the number of classes. The subsequent lines list the data for each class, one after the other.

Each set of class data begins with a line indicating the number of test scores recorded for the class followed by a separate line for each score listing the student's first name and their score. Students with multiple scores will appear in the list multiple times.

Output

The program prints a set of output for each class which includes a line indicating the number of students ("X students") followed by an alphabetized list of student names and letter grades.

Note, when determining letter grades, truncate student averages to integers before looking up corresponding letter grades. For instance, an average of 72.333 truncates to 72 which is a D.

Example Input File

```
2
3
James 100
Marc 50
Tim 75
7
Tim 75
Marc 50
James 100
Marc 110
Tim 0
Sparky 5
James 0
```

Example Output To Screen

```
3 students
James A
Marc F
Tim C
4 students
James F
Marc C
Sparky F
Tim F
```

Problem #2: Bowling for Dummies

Program Name: bowling.java   Input File: bowling.dat

Bowling is a fun game for all ages. The object of the game, like so many others, is to get the highest score possible. Points are scored by rolling a ball down a lane and knocking down ten pins that are arranged in a triangular pattern.

A game is scored by summing the scores for each of ten *frames*, where each frame a player is given up to two rolls to knock down the pins. If a player knocks down all the pins on the first roll of a frame, the frame is scored as a strike and marked with an 'X'. A strike is worth ten points plus one point for each pin knocked down in the next two rolls. If a player knocks down all remaining pins in the second roll of a frame, the frame is scored as a spare and marked with a '/'. A spare is worth ten points plus one point for each pin knocked down in the next roll. If the player does not knock down all the pins in the two rolls for a frame, the score for the frame is the total number of pins knocked down for that frame.

The tenth frame is an exception to the above rules and its score is the total number of pins knocked down. Unlike the other frames, if a player scores a strike on his first roll for the tenth frame, he is allowed two more rolls for the frame, or if he scores a spare on his first two rolls of the frame, he is allowed one more roll for the frame.

Given the above scoring method, the highest possible score for a bowling game is 300.

Input

The first line will be a single integer, *n*, indicating the number of games to be scored.

Each of the next *n* lines will represent a single game and will consist of all the rolls for that game. The outcome of each roll will be indicated with an integer representing the number of pins knocked down, a '/' for a spare, or a 'X' for a strike.

Output

Output will consist of *n* lines, where each line is the score of a bowling game from the input. Game scores should be listed in the order of the games in the input.

Example Input File

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

Example Output To Screen

```
183
121
227
```

Problem #3: Function Finder Fun

Program Name: function.java   Input File: function.dat

Sometimes in programming it's useful to know what functions are called by a snippet of code. You want to write a program that will do just that.

Input

Input will be a snippet of code, 1-100 lines in length.

Output

Output will be the function names, in the order that they occur in the code, each on their own line. For the purpose of this problem, a function name will be a sequence of alphabetic [a-zA-Z] characters immediately to left of a '(' as long as the name is not one of the following reserved words:

if,while,for,switch,return

Function names should only be output the first time they are encountered in the code.

Example Input File

```
int x,y;
int z=(3 + 2) * 5;

System.out.println("Test");

for(x = 0; x < 20; x++)
   for(y = 0; y < 30; y++)
      if(y == 1)
         System.out.println(x + "+" + y + "=" + (x + y));

while(z <= 25)
{
   foo(z);
   parse(z + 3);
   z++;
}

return(x * y * z);
```

Example Output To Screen

```
println
foo
parse
```

Problem #4: Radiant Primes

Program Name: radiant.java **Input File: radiant.dat**

In the course of finding radiant primes an interesting algorithm is used. The algorithm is the following:

Given a base 10 integer X, and a base Y:
1.Convert X to base Y.
2. Reverse the digits of the result from step 1.
3. Convert the result from step 2 to base 10.

It is of interest whether the result of step 3 is prime or non-prime when studying radiant primes. Remember that an integer greater than zero is prime if it has exactly one divisor other than 1.

Here is an example:
17 3
17 in base 3 is 122
Reversed is 221
which is 25 in base 10
which is non-prime

Another example:
2 9
2 in base 9 is 2
Reversed is 2
which is 2 in base 10
which is prime

Input
The first line of the input will be a single integer, N, indicating the number of data sets.
Each of the next N lines will be X and Y, as indicated in the algorithm. The X and Y will be separated by a single space. 2 <= X <= 1000 and 2 <= Y <= 9.

Output
Output for each data set will be a single line with the word "prime" or "non-prime", depending on the result when the algorithm is applied to X and Y.

Example Input File
```
3
11 2
17 3
2 9
```

Example Output To Screen
```
prime
non-prime
prime
```

Problem #5: Scheduling Nightmare on Elm Street

Program Name: dvr.java **Input File: dvr.dat**

You've been having some nasty nightmares lately and plan to stay up as late as possible tonight watching television. You know the quality of late-night programming is sub-par, so you're hoping your DVR (digital video recorder) will record some television shows ahead of time. Unfortunately, your DVR is having problems with its scheduling algorithm, so you have to write a new one for it. You recall the way your DVR's scheduling algorithm works is to record shows in a priority list at the earliest possible time. If two or more shows have conflicting recording times, the show with the highest priority is recorded. Each show is recorded at most once.

Input

There will be only one data set for this problem.
The first line of the data set will be in the format `"X Y"`, where 1 <= X <= 10 and 1 <= Y <= 20. X is the number of shows in your priority list. Y is the number of entries in the TV guide for the shows on your priority list.
The next X lines will be the priority list for the shows you wish to watch. Each show will be on its own line, ordered by priority (the first show has the highest priority).
The next Y lines represent the unsorted TV guide listings for shows on your priority list and will be in the format `"<showtime> <show title>"`. Showtimes will be in military format (`"0000"`-`"2330"`) and will always end in `"00"` or `"30"`. All shows will be thirty minutes long.

Output

Output will be the recorded shows, in the order that they are recorded. Each recorded show will be on a separate line.

Example Input File

```
7 12
Friends
Rugrats
Wheel of Fortune
Channel 9 News
The Simpsons
George of the Jungle
Everybody Loves Raymond
2000 Friends
0800 Rugrats
1600 The Simpsons
1630 The Simpsons
1700 The Simpsons
1730 The Simpsons
1600 Wheel of Fortune
1500 Channel 9 News
2000 George of the Jungle
2000 Everybody Loves Raymond
2100 Everybody Loves Raymond
1630 George of the Jungle
```

Example Output To Screen

```
Rugrats
Channel 9 News
Wheel of Fortune
The Simpsons
Friends
Everybody Loves Raymond
```

Problem #6: Bust ‘A Move

Program Name: bust.java   Input File: bust.dat

The classic video game, Bust ‘A Move, is based on the ability of the player to form groups of balloons that are the same color. When these groups are formed, all the balloons in the group suddenly burst, leaving a much smaller set of balloons standing between the player and the end of the level.

A computer program that plays this type of game needs to be able to distinguish groups of similar objects.

For this problem, write a program that can determine the number of distinct groups of balloons of the same color on a 5x10 game board.

There are four colors of balloons:

‘R’ - Red
‘B’ - Blue
‘G’ - Green
‘Y’ - Yellow
‘.’ (period) - no balloon

Input

The input will consist of multiple game boards. The first line of the input indicates the number of game boards to be processed. The rest of the file contains the game boards. Each board is five lines high and ten columns wide.

Output

Print one line of output for each game board: X groups. Where X is the number of distinct, contiguous groups of balloons on the game board.

Two balloons are considered to be in the same group if they are vertically or horizontally adjacent and are the same color.

Note that periods denote empty space and do not form groups.

Example Input File

```
3
..........
.RR.......
.RR.......
..........
..........
..........
.R.R.BBB..
.RRR..B...
.R.R.BBB..
..........
RRRRRRRRRR
BBBBBBBBBR
RRRRRRRRRR
RGGGGGGGGG
RRRRRRRRRR
```

Example Output To Screen

```
1 groups
2 groups
3 groups
```

Problem #7: Poly's Nomials

Program Name: poly.java   Input File: poly.dat

Poly is great at programming and good at math. One of her friends is good at neither and has lots of algebra homework that Poly has to help her with. In order to give herself some free time from her needy friend, Poly has decided to write a program that will put polynomials in canonical form. They are input with the terms out of order, and her program orders the terms and displays the resulting polynomial. Hopefully Poly's friend can figure out how to run it!

For this problem, the standard form of a polynomial is:

```
    [-] Ax^9 +/- Bx^8... +/- Hx^2 +/- Ix +/- J
```

A-J are integer coefficients ranging from -1000 to 1000, and the `+/-` indicates the sign of the coefficient.
For example:

```
    - 2x^9 + x^8 - 1
```
or
```
    1000x^9 - x^3
```

If a coefficient is 1 or -1 the number itself is omitted unless it is used in the constant term. (e.g., "`- x^9`" and "`-1`" are valid terms, but "`- 1x^9`" is not).
Terms appear in descending order by exponents of x, with 9 being the maximum exponent.
Exponents are only displayed if they are greater than 1. (i.e., "`5x^1`" is incorrect, but "`5x`" is correct)
Many terms may have a coefficient of zero (and are not shown), but there will be at least one term with a non-zero coefficient in every polynomial.
At most one term of each degree will be given in any one polynomial (i.e., at most one with x^9, at most one with x^8, etc.)
There are single spaces separating the + and – operators from the terms.
The leading term will only have an operator displayed if its coefficient is negative (i.e., a + is otherwise implied)

Input
The first line indicates the number of polynomials to be converted. Each subsequent line contains one polynomial.

Output
For each polynomial in the input, output the polynomial in standard form on its own line.

Example Input File
```
3
7 + 4x^8 - 34x^2 - x
x^8
- 17 + x^8 - 9x^7 - 243x^9 - x^5
```

Example Output To Screen
```
4x^8 - 34x^2 - x + 7
x^8
- 243x^9 + x^8 - 9x^7 - x^5 - 17
```

Problem #8: Who's the Boss?

Program Name: boss.java   Input File: boss.dat

As a successful programmer for MegaCorp, you are constantly bombarded by requests from myriads of different people. Until recently, you spent most of your time trying to figure out whether or not these requests merited attention. Luckily, MegaCorp just published a new organization chart showing the management structure for every employee, and all you have to do is write a program that will determine if someone is your boss (or your boss's boss, or boss's boss's boss, etc.).

The basic goal for the program is this: Given an organization chart, determine if someone has to obey an order from another.

Input

The input is divided into two sections. The first section is the organization chart for MegaCorp. The second section is a list of name pairs.

MegaCorp's organization chart is given as a list of teams, and the first line of input indicates the number of teams. This is followed by one line for each team. A team line consists of the total number of minions, the boss name, and the minion names (for example, "`3 boss minionA minionB minionC`").

Note, the organization chart only lists first names, and each name uniquely identifies a person. A person will appear as a minion in any team at most once, and there will be no circular relationships (i.e., a boss will never report directly or indirectly to one of his minions).

The list of name pairs begins with a single line indicating the number of name pairs. This is followed by the name pairs, each on its own line. In each pair, the first person wants to know if they have to follow instructions given by the second person. In other words, they want to know if the second person is their boss, their boss's boss, etc.

A name pair will always contain names from the organization chart and will never contain the same name twice.

Output

For each name pair (<first> <second>) in the input, determine if the first person must obey the second and print either:

  <first>: Sure <second>, I'll get right on it.

or

  <first>: No <second>, I don't have time to do your work and mine.

Example Input File

```
4
2 BigBoss John Fred
3 John Wilson Stoker Bubba
4 Wilson James Marc Tim Alan
2 Fred Wilma Dino
3
Alan Dino
Alan BigBoss
Wilson James
```

Example Output To Screen

```
Alan: No Dino, I don't have time to do your work and mine.
Alan: Sure BigBoss, I'll get right on it.
Wilson: No James, I don't have time to do your work and mine.
```

Problem #9: Enemy at the Gates

Program Name: gates.java   Input File: gates.dat

You are building a circuit which consists of a hierarchy of logic gates which take a series of 2^n bits as input. The bits are processed in stages. The first stage pairs all the bits from the input and applies a logic operation to each pair, leaving 2^{n-1} bits for the next level. The process is repeated at each subsequent level until only 1 bit remains. You have decided to write a program to aid you in the design of this circuit that will determine for a given set of input bits and a given logic gate layout, what the output bit will be.

Below is a description of each type of logic gate that your circuit can consist of:

AND GATE - Outputs a 1 if both bits are 1, outputs 0 otherwise.

OR GATE - Outputs a 1 if at least one of the bits is 1, outputs 0 otherwise.

XOR GATE - Outputs 0 if both bits are equal, outputs 1 otherwise.

Below is a schematic of a possible circuit that could be input into your program. It corresponds to the second data set in the sample input:

```
1 -
  A - 0 -
0 -      |
        X|- 0 -
0 -      |     |
  A - 0 -      |
1 -            |
              X| - 0
1 -            |
  R - 1 -      |
1 -      |     |
        A|- 0 -
0 -      |
  R - 0 -
0 -
```

Input

The first line of input will contain an integer number indicating how many data sets are to be processed. The remaining lines will contain the data sets. Each data set will consist of three lines formatted according to the following descriptions:

Bit Count - An integer corresponding to the number of bits being fed into the circuit. This integer will be a number between 2 and 32 and will be a power of 2.

Bit List - A space separated list of bits (either 0 or 1) that will be fed into the circuit.

Gate List - A space separated list of logic gates. The first $n/2^1$ gates constitute the first stage and operate directly on the input bits, the next $n/2^2$ gates constitute the second stage and operate on the results from the first stage, and so on for the other stages until only one gate remains. (Note, it is possible the there is only one stage) Each character in the list will either be 'A', 'R', or 'X' (corresponding to AND, OR, or XOR gates respectively). There will be exactly n-1 logic gates (where n is the number of bits read in).

Output

The output will consist of the bit (0 or 1) resulting from the logic gate operations on the input bits. Each pair of bits will be fed into the first $n/2^1$ gates, and the results of each of those operations will be fed into the next $n/2^2$ gates, and so on until the result is reduced to a single bit (0 or 1).

Example Input File

```
2
2
1 1
A
8
1 0 0 1 1 1 0 0
A A R R X A X
```

Example Output To Screen

```
1
0
```

Problem #10: Let's Make a Meal

Program Name: meal.java **Input File: meal.dat**

You are burning the midnight oil while writing a program for school. Your mom tells you that you need to eat something and that it needs to be a well-balanced meal. To make sure that your mom is happy, you write a program that determines if a list of foods constitutes a well balanced meal given the following list of basic programming food groups (She never said the food groups had to be healthy ones, now did she?):

Caffeine Group	Soda, Coffee
Sugar Group	Cake, Candy
Salt Group	Chips, Popcorn
Fat Group	Burger, Pizza

Input

The first line of input will contain an integer corresponding to the number of data sets to be read in.

The subsequent lines contain the data sets, each of which will be formatted according to the following description:

Food Count – A single line containing the number of food items, *n*, on the Food List. This will be a number between 4 and 8 (inclusive).

Food List – The next *n* lines will each list a single food item. If the food name contains the exact name (case insensitive) of a food category listed in a food group, then it is a member of that food group. A food item can be a member of multiple food groups. The snack list will be deemed well balanced if it contains at least 1 food item from each of the 4 listed food groups and no more than 2 food items from any 1 of the listed food groups.

Output

If the Food List is well balanced according to the input description, then the following will be printed:

"Programmer Fuel"

If the Food List is not well balanced according to the input description, then the following will be printed:

"What would your mom say?"

Example Input File

```
2
4
Carrots
Orange Soda
Chocolate Cake
Hamburger
6
Red Soda
Coffee
Chocolate Candy
Potato Chips
Cheeseburger
Buttered Popcorn
```

Example Output To Screen

```
What would your mom say?
Programmer Fuel
```