University Interscholastic League

Computer Science Competition

2006 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.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Point Values and Names of Problems

Number	**Name**	**Point Value**
Problem 1	What's The Frequency, Kenneth?	60
Problem 2	What's For Dinner?	60
Problem 3	Going The Distance	60
Problem 4	Look Out!	60
Problem 5	Amazing Maize Mazes	60
Problem 6	Passing Notes	60
Problem 7	I8 Your Battleship!	60
Problem 8	Skyline	60
Problem 9	Suffix Sorting	60
Problem 10	Triage Treatment	60
Problem 11	Give Me A Vowel	60
Problem 12	I Want To Win	60
Total		**720**

Problem 1 **60 Points**

What's The Frequency, Kenneth?

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

A good approach to cracking replacement cyphers is frequency analysis. It uses known frequencies of letters in normal text to help decrypt encoded messages. However, before this approach can be employed, the cryptologist has to have a very good idea what the expected frequencies of letters are in normal text.

Write a program that will output the frequencies of letters occurring in the input text.

Input

The first line of input will contain a single integer, *n*, indicating the number of lines in the text to be examined. The following *n* lines are to be examined by the program to determine the frequencies of the letters used.

Output

There will be 26 lines of output, one for each alphabetic character, beginning with 'a' and ending with 'z'. Each line will contain the letter followed by the number of times it occurred in the input text. Ignore the case of the characters in the input. For example, both 'E' and 'e' will count toward the frequency of 'e'.

Example Input File

```
3
The quick brown fox jumps over the lazy dog
Hello
Goodbye
```

Example Output To Screen

```
a 1
b 2
c 1
d 2
e 5
f 1
g 2
h 3
i 1
j 1
k 1
l 3
m 1
n 1
o 7
p 1
q 1
r 2
s 1
t 2
u 2
v 1
w 1
x 1
y 2
z 1
```

Problem 2 | **60 Points**

What's For Dinner?

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

Farmer Jane raises dogs and rabbits, feeding the latter carrots, and feeding the former the latter. She has noticed that each day, each rabbit will go one at a time to the carrot field and eat three carrots (or however many carrots are left if there are less than three). Any rabbit that did not eat three carrots will die of starvation. Then, each dog will go one at a time to the rabbit pen and eat two (or however many rabbits are left if there are less than two) of the live rabbits (they do not wish to eat dead, malnourished rabbits). Any dog that did not eat two rabbits will die of starvation. Farmer Jane is interested in knowing how many dogs, rabbits, and carrots she will have after a certain number of days given these conditions.

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 a single line:

1. A line in the format "*a b c d*", where *a* (0-1000) is the starting number of dogs, *b* (0-1000) is the starting number of rabbits, *c* (0-1000) is the starting number of carrots, and *d* (1-100) is the number of days to process.

Output

For each data set in the input display the following:

1. A single line, "*x y z*" where, given the conditions described, *x* is the number of dogs at the end of the *d*th day, *y* is the number of rabbits at the end of the *d*th day, and *z* is the number of carrots at the end of the *d*th day.

Example Input File

```
3
10 10 6 1
10 100 900 4
1 2 2 1
```

Example Output To Screen

```
1 0 0
10 20 60
0 0 0
```

Problem 3 **60 Points**

Going The Distance

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

Choosing a long-distance plan can be quite complicated. Many plans give callers a fixed number of minutes for a flat monthly fee, with additional minutes being charged at a premium rate. This gives callers an incentive to choose a plan wisely to avoid paying too much for their service.

Write a program that can determine the best long-distance plan for a given caller given the caller's number of minutes used per month and the information for all available plans.

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 begin with a line containing two integers, *p* and *c*, where *p* indicates the number of calling plans available and *c* indicates the number of callers that need to have their optimal plan calculated.

The next *p* lines of the data set list details for each of the long distance plans, and are formatted "Plan <pname> <minutes> $<prmonth> $<additional>" where <pname> is the name of the plan, <minutes> is the number of minutes included per month for the <prmonth> monthly fee, and <additional> is the cost for each minute used in excess of <minutes>.

The next *c* lines of the data set list details for each caller that must choose between the plans and are formatted "<cname> <minutes>", where <cname> is the customer's name, and <minutes> is the number of long-distance minutes that this person uses each month.

Output

For each data set in the input output the header line:

Data set #*X*

where *X* is 1 for the first data set, 2 for the second, etc.
Then, for each caller in the data set, output the message:

<cname> should choose plan <pname> for $<mcost> per month.

Where <cname> and <pname> are caller and plan names from the data set and <mcost> is the lowest monthly cost the caller can achieve from the available plans when using the specified number of minutes. You can safely assume that there will always be a single plan with the lowest cost (i.e., no ties).

Display the cost to 2 decimal places.

Example Input File
```
2
2 2
Plan A 100 $10.00 $0.15
Plan B 200 $20.00 $0.15
Ralph 150
Lauren 250
3 3
Plan UnlimitedPlus 0 $29.95 $0.00
Plan OneRate 0 $3.95 $0.07
Plan TalkTime 30 $3.50 $0.10
Julius 1000
Caesar 200
Chavez 10
```
Example Output To Screen
```
Data set #1
Ralph should choose plan A for $17.50 per month.
Lauren should choose plan B for $27.50 per month.
Data set #2
Julius should choose plan UnlimitedPlus for $29.95 per month.
Caesar should choose plan OneRate for $17.95 per month.
Chavez should choose plan TalkTime for $3.50 per month.
```

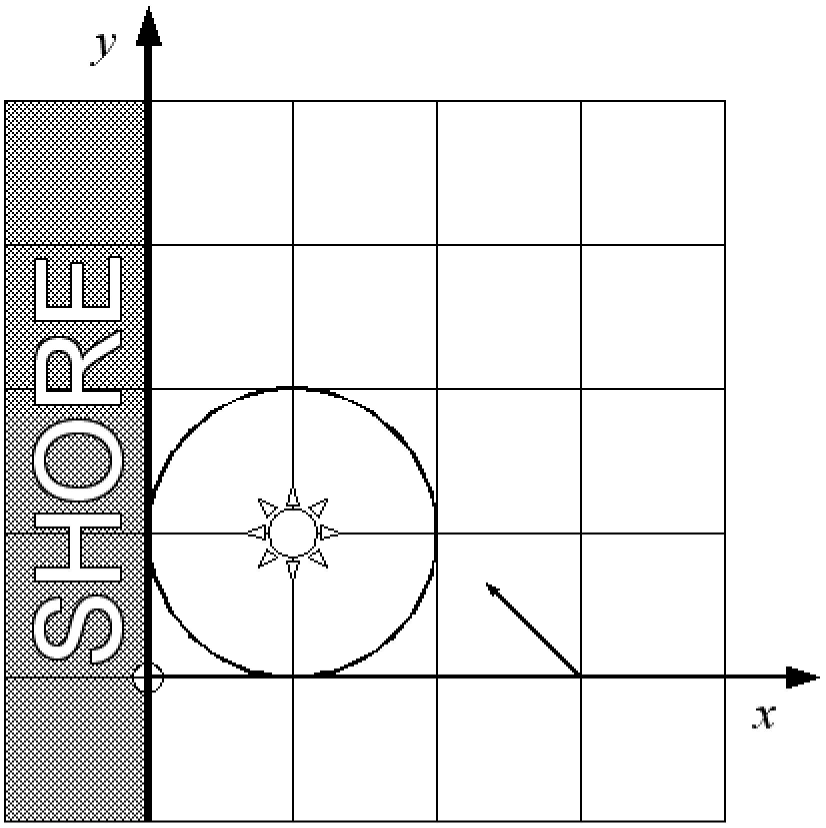
Problem 4

Look Out!

60 Points

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

You are piloting a boat at night through unknown waters. There is only one lighthouse, and to survive you have to find the lighthouse before you hit the shore. Since you don't know exactly where the shore is, you decide to just steer straight and let fate decide the outcome.



Write a program that, given the location of the lighthouse and the path of the boat, will determine whether or not the ship lands safely, sunders its hull on the shoals, or is lost at sea.

The entire scenario takes place in the *x,y* coordinate plane, with the positive *x*-axis pointing east and the positive *y*-axis pointing north. The shore line is the *y*-axis ($x = 0$), with the ocean on the positive *x* side. The lighthouse is in a stationary position, given as an (*x,y*) coordinate pair, and its light is visible to anyone at or within *r* nautical miles. Your boat has a starting position, given as another (*x,y*) coordinate pair, and a heading that is given in degrees. The heading is measured as an angle counter-clockwise from the positive *x*-axis (i.e., 0 degrees is east, 90 degrees is north, 180 degrees is west, and 270 degrees is south).

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 consists of a single line with six integers, separated by spaces (*lx ly r bx by h*). The pair (*lx*, *ly*) is the position of the lighthouse, and *r* is the distance from the lighthouse at which the light can be seen. The pair (*bx*, *by*) is the starting position of the boat, and *h* is the boat's heading.

The boat's starting position will always be in the water (i.e., $bx > 0$), but the lighthouse can be anywhere.

Output
For each data set in the input print a single line depending on which of the three possible outcomes applies. The three possible outcomes are:

1. The boat never hits the shore and never sees the light from the lighthouse. Print “Lost at sea”.
2. The boat hits the shore before, if ever, seeing the lighthouse. Print “Look out!”.
3. The boat sees the lighthouse before, if ever, hitting the shore. Print “Saved by the lighthouse”.

If the boat sees the lighthouse at the same time that it hits the shore, it's too late to change course and outcome #2 applies (“Look out!”).

The diagram shown at the start of the problem depicts the third data set.

Example Input File
```
5
0 0 5 10 0 180
0 0 5 10 0 270
1 1 1 3 0 135
0 1 1 3 0 135
0 0 2 1 3 225
```
Example Output To Screen
```
Saved by the lighthouse
Lost at sea
Saved by the lighthouse
Look out!
Look out!
```

Problem 5 | **60 Points**

Amazing Maize Mazes

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

Atty loves to travel across America and explore mazes cut into cornfields, or maize mazes. Help her find her way through each of these mazes.

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 two parts:

1. A line in the format "*w h*", where *w* (1-50) is the width of the maze and *h* (1-50) is the height of the maze.
2. The next *h* lines will each contain *w* characters, with each character being one of the following:
   - '@' -- This represents Atty's starting and ending locations. There will be exactly two of these per maze.
   - '#' -- This represents corn and is impassable by Atty.
   - '.' -- This represents empty space through which Atty may pass.

Output

For each data set in the input display the following:

1. A maze exactly as read in, except with the shortest path from the starting location to the ending location marked, by replacing all '.'s along the path with '@'s. There will be one and only one shortest path per maze.

Example Input File

```
3
10 7
##@#######
#.....##@#
#.#.#.##.#
#.#.#.##.#
#.#.####.#
#........#
##########
2 1
@@
5 5
@...#
#.#..
...#.
.#..#
..#.@
```

Example Output To Screen

```
##@#######
#.@@..##@#
#.#@#.##@#
#.#@#.##@#
#.#@####@#
#..@@@@@@#
##########
@@
@@..#
#@#..
.@@#.
.#@@#
..#@@
```

Problem 6 | **60 Points**

Passing Notes

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

A summer camp has recently decided to try a new ice breaker activity to help get its 100 attendees to know each other better. It involves breaking the campers up into groups to play games, then changing the groups and playing another game, and continuing to change groups and play games until the camp counselors are ready to do something else.

Two of the campers are sweethearts, and one wants to pass a note to the other as quickly as possible without the counselors noticing. A person with a note can pass it to anyone in their current group, and the person holding the note when groups are changed will carry the note to their new group. Groups change every 5 minutes, so write a program that can determine the minimum amount of time required to pass the note between the two sweethearts.

Fortunately, you know the algorithm that determines the way the counselors divide up the campers. First, each camper is assigned a number from 1 to 100. Then the counselors generate a list of numbers that will be used to divide up the groups. They refer to the first number to create the first set of groups by putting together campers whose assigned number has the same remainder when divided by the number from the list. For instance, if the first number of the counselors' list was 3, then there would be 3 groups. One group would contain all campers whose assigned numbers divide 3 evenly (3, 6, 9, 12, etc.), one group would be for those where the remainder was 1 (i.e., 1, 4, 7, 10, etc.), and the third group would be for those where the remainder was 2 (i.e., 2, 5, 8, 11, etc.). After 5 minutes, the counselors look at the next number on their list and then have the campers form new groups.

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 be a single line containing 12 positive integers. The first indicates the number assigned to the note writer, and the second is the number of his/her sweetheart. The remaining 10 integers make up the counselor's list used to create the groups.

Output
For each data set in the input, print a single line indicating the minimum number of minutes from the time of the first grouping to pass the note to the writer's sweetheart. Format the output as, “x minutes”.

You can assume that the note will always, eventually, reach its destination.

Example Input File
```
3
1 100 10 3 5 5 5 5 5 5 5 5
18 27 1 2 3 4 5 6 7 8 9 10
4 71 50 10 2 8 16 72 70 100 3 2
```
Example Output To Screen
```
5 minutes
0 minutes
40 minutes
```

Problem 7 **60 Points**

I8 Your Battleship!

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

Playing the game of battleship requires players to place ships on a rectangular grid hidden from the other player. Players then take turns taking 'shots' at their opponent's board by calling out grid positions (e.g., E3) and having the opponent indicate whether the shot was a hit or miss.

Write a program that will read in a given starting position for one player and then process the shots made by that player's opponent. The program will indicate whether each shot is a miss or a hit. If it is the last hit on a given ship, it will print a special message indicating that the given ship has been sunk.

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 three parts:

1. Player A's game board. This consists of 10 lines of 10 characters each. Each character either shows empty ocean (represented by a period) or a piece of a ship. There is exactly one of each type of ship on the board, and each is placed either horizontally within a single row or vertically within a single column. Different ships are represented by different capital letters, as shown below along with the ship lengths:
   'C' – carrier (5 units)
   'B' – battleship (4 units)
   'D' – destroyer (3 units)
   'S' – submarine (3 units)
   'P' – patrol boat (2 units)
2. A line containing a single integer, *m*, indicating the number of shots Player B will make (less than 20).
3. A line containing a space-separated list of all of Player B's shots. Shots are of the format <row><col> where <row> is A, B, C, etc. starting from the top, and columns are 1, 2, 3, etc. starting from the left. For example, if Player B makes 3 shots, this line might look like “A5 D2 G7”.

Output

For each data set in the input display the following:

1. A single line, “Data Set #X” where X is 1 for the first data set, 2 for the second, etc.
2. The list of Player B's shots, one per line, followed by the result of that shot. If the shot hits the last remaining piece of one of Player A's ships, the result of the shot is “sank my <stype>!” where <stype> is the type of ship. If the shot hits a ship which still has some portion that has not been hit, the result of the shot is “hit!”. Otherwise, the result of the shot is “miss”.

Example Input File

```
2
..........
..........
..CCCCCB..
..D....B..
..D....B..
..DSSSPB..
......P...
..........
..........
..........
3
A5 D2 G7
....C.....
....C.....
....C.....
....C.....
....CBBBB.
..DDDS....
.....S....
.....S....
P.........
P.........
14
A1 A10 J10 J1 I1 E5 E4 E6 F6 G6 H6 F5 F4 F3
```

Example Output To Screen

```
Data Set #1
A5 miss
D2 miss
G7 hit!
Data Set #2
A1 miss
A10 miss
J10 miss
J1 hit!
I1 sank my patrol boat!
E5 hit!
E4 miss
E6 hit!
F6 hit!
G6 hit!
H6 sank my submarine!
F5 hit!
F4 hit!
F3 sank my destroyer!
```

Problem 8

Skyline

60 Points

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

Given the height of skyscrapers in a city's downtown area, construct the view that an observer to the west would have of those buildings.

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 begins with a line containing a single integer, *m* (0 <= *m* <= 9), indicating the size of the downtown area (*m* x *m*). The next *m* lines each contain *m* integers from 0 to 9 representing the heights of buildings. Viewed as a whole, the *m* x *m* array of integers represents an overhead topological view of the city, with north at the top.

Output

For each data set in the input display the following:

1. A single line, “Data Set #X” where X is 1 for the first data set, 2 for the second, etc.
2. The view of the downtown skyline as seen from an observer standing to the west of town. Instead of using true perspective (where buildings would look smaller in the distance), please show a simple projection of the buildings. Each visible building should be represented with an integer from 1 to 9 (inclusive) indicating the distance of the building from the observer, with 1 being the closest possible building (i.e., one that's in the leftmost column of the overhead view). The output view should always be as high as the tallest building, with periods used to fill in space above the tallest buildings.

Example Input File

```
2
5
00005
00004
50003
00002
00001
3
189
015
001
```

Example Output To Screen

```
Data Set #1
5.1..
551..
551..
5515.
55155
Data Set #2
3..
2..
2..
2..
23.
23.
23.
23.
123
```

Problem 9

Suffix Sorting

60 Points

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

At times it can be useful to sort lists of words in non-traditional ways. For instance, when looking for words that share a common suffix, it is useful to perform a sort by comparing the last letters first, then the next-to-last letters, and so forth. That is what your program will do.

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 begins with a line containing a single integer, *m*, indicating the number of words in the data set. The following *m* lines each contain a single word of at most 20 alphabetic characters.

Output

For each data set in the input display the following:

1. A single line, "Data Set #X" where X is 1 for the first data set, 2 for the second, etc.
2. The sorted list of words from the data set. Words are sorted by comparing the last characters, then the next-to-last characters, etc. Characters should be sorted in ascending order (a, b, c, etc.) without regard for case.

Example Input File

```
2
3
bus
calculus
abacus
5
move
five
JIVE
hives
love
```

Example Output To Screen

```
Data Set #1
bus
abacus
calculus
Data Set #2
five
JIVE
love
move
hives
```

Problem 10 **60 Points**

Triage Treatment

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

Triage is a process for sorting injured patients into groups based on their need for or likely benefit from immediate medical treatment. At most veterinary clinics, veterinarians will interrupt treatment of a patient if another patient with higher medical priority arrives. Only after higher medical priority patients have been tended to will resources be once again diverted to lower medical priority patients.

Write a program for a veterinary clinic that will give an estimated time for when all patients will have been successfully treated.

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 two parts:

1. A line containing a single integer, *m*, indicating the number of patients.
2. Each of the next *m* lines will correspond with a single patient and be in the format "*x y z*", where:
   - *x* is the time the patient arrives at the clinic, specified by the number of minutes (0-480) that have elapsed since the clinic opened. Patients are listed in order of arrival, so the *x* value for each patient is at least as high as for the previous patient.
   - *y* is the medical priority (1-5) of the patient, with 1 being the highest medical priority.
   - *z* is the number of uninterrupted minutes (1-480) of treatment necessary to tend to the patient.

Output

For each data set in the input display the following:

1. A single line "*a*", where *a* is the number of minutes that have elapsed since the clinic opened when all patients have been successfully treated. Only one patient can be treated at a time (it is a small clinic) and treatment for a patient can only begin after it has arrived at the clinic. Treatments are begun as soon as possible. If two or more patients are at the clinic awaiting treatment, the one with highest medical priority is treated first. If two or more have the same highest medical priority, then the one (of those with the highest medical priority) with the shortest treatment time is treated first. The only time treatment may be interrupted is when a higher medical priority patient arrives. In this case, the treatment for the lower medical priority patient is stopped and treatment for the highest medical priority patient begins. For a patient to be successfully treated, its treatment must be uninterrupted and last the number of minutes specified in the input.

Example Input File

```
3
3
0 5 20
19 4 30
48 3 60
7
60 1 30
75 1 60
80 2 10
80 3 5
80 2 5
155 1 10
166 2 5
3
0 3 10
1 3 5
2 4 1
```

Example Output To Screen

```
158
185
16
```

Problem 11 **60 Points**

Give Me A Vowel

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

Write a program that can determine if a given word begins with a vowel or a consonant.

Input

The first line of input will contain a single integer, *n*, indicating the number of words to process. The following *n* lines each contain a single word which should be processed separately.

Output

For each word in the input, print the following message, “The word '<word>' begins with a <letter-type>.” In place of <word>, print the input word. In place of <letter-type> print “vowel” if the word begins with one of the 10 characters AEIOUaeiou, otherwise print “consonant”.

Example Input File

```
7
Give
Me
A
Vowel
or
a
consonant
```

Example Output To Screen

```
The word 'Give' begins with a consonant.
The word 'Me' begins with a consonant.
The word 'A' begins with a vowel.
The word 'Vowel' begins with a consonant.
The word 'or' begins with a vowel.
The word 'a' begins with a vowel.
The word 'consonant' begins with a consonant.
```

Problem 12

I Want To Win

60 Points

Program Name: win.java   Input File: [none]

To ensure that all teams in this competition have the desire to win, each will need to write a program that states their desire to win an obscene number of times.

Input
There is no input for this problem.

Output
Output the sentence "I want to win!" exactly 27356 times, with each instance on its own line.

Example Input File
[none]

Example Output To Screen
[the output is too long to show in its entirety, but looks something like this]

```
I want to win!
I want to win!
I want to win!
I want to win!
.
.
.
I want to win!
```
[27356th line]