University Interscholastic League

Computer Science Competition

2005 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 72 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	Million Monkey Mayhem	72
Problem 2	Horse Power	72
Problem 3	Word Search	72
Problem 4	Bullet Time	72
Problem 5	Tales From The Crypt	72
Problem 6	What's Not On The Menu	72
Problem 7	Word Wrapper	72
Problem 8	Dumb Waiter	72
Problem 9	Lather, Rinse, Repeat	72
Problem 10	Amazing Mouse	72
Total		**720**

Problem 1 | **72 Points**

Million Monkey Mayhem

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

We've all heard the expression that "Given enough time, a million monkeys typing on a million typewriters would eventually produce the works of Shakespeare" (which severely underestimates the magnitude of the problem). But let's assume we have semi-literate monkeys and we only wish them to type a short sentence. We'll use a pseudorandom number generator to simulate a monkey banging away on a typewriter.

Java conveniently supplies a pseudorandom number generator using the Random class. The only problem is that we need to randomly generate characters while the Random class generates signed integers. Use the following formula to map a signed integer, *s*, to a printable ASCII character value:

$$\textit{ASCII Value} = (| s | \bmod 26) + 97$$

This will generate an ASCII value in the range [97,122] which represents a lower-case letter. If we generate 10 random integers (using the nextInt() method) we can use the above formula to produce a pseudorandom string of ten letters.

Input

The first line of input will contain a single integer *n* indicating the number of data sets. Each data set will consist of a single line containing an integer that is the seed value to use for the pseudorandom number generator.

Output

The output for each data set will be the string of ten characters produced by the pseudorandom number generator along with the formula given above.

Example Input File

```
4
262
82
1
8836
```

Example Output To Screen

```
sctozzonyj
copdbmfmza
napsywpddc
rfzkxwthee
```

Problem 2 | **72 Points**

Horse Power

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

In a study published by Drs. Pagan and Hintz in the *Proceedings of the Equine Nutrition and Physiology Society*, a formula was developed for calculating calorie expenditure in horses while exercising:

$Y = (e^{3.20 + .0065s})(x)(z)$; where s is the speed in meters/minute, x equals the horse's weight in kilograms, z equals the amount of time exercising, and Y equals the calories expended.

Example:

So if a 400kg horse traveled 40,000 meters in 250 minutes, the speed would be 160 meters/minute, and the formula would look like:

$Y = (e^{3.20 + .0065(160)})(400)(250)$
$Y = 6940785$ (rounded to nearest integer)

Input

The first line of input will contain a single integer *n* indicating the number of data sets. Each data set will consist of a single line containing the horse's weight (in kg), the distance the horse traveled (in meters), and the amount of time it took to travel that distance (in minutes). These values will be separated by a single space.

Output

The output for each data set will be the amount of calories expended by the horse, given the formula above. Calories should be rounded to the nearest integer.

Example Input File

```
3
400 40000 250
500 30000 260
680 2000 2
```

Example Output To Screen

```
6940785
6751598
22191946
```

Problem 3 | **72 Points**

Word Search

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

To kill some time before the contest, you bought a book of word search puzzles. It's a tough one because words can be spelled in any of 8 directions (up, down, left, right, and all 4 diagonal directions), so after a while you get bored and write a program to find the words for you.

Input

The first line will contain a single integer *n* indicating the number of data sets. Each data set will consist of 2 components:

1. A 10x10 matrix of lowercase characters to be searched.
2. The next 10 lines will consist of lowercase words (1-10 characters).

Output

For each word, print the following if it was found in the matrix:

"<word> was found"

or print the following if the word was not found in the matrix:

"<word> was not found"

Example Input File

```
1
abcdefghij
klmnullrst
uvwxyzabcd
efgoljklmn
opqrsruvwx
ymbiadefgh
ijkltnopqr
stuvwndzab
cdefghijkl
mnopqrstuv
null
programmer
xor
rand
rox
z
perl
ibm
void
int
```

Example Output To Screen

```
null was found
programmer was not found
xor was found
rand was found
rox was found
z was found
perl was found
ibm was found
void was not found
int was found
```

Problem 4		72 Points

Bullet Time

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

Given the starting position of a bullet and its position after one second in flight, give the position of the bullet after 5 seconds in flight.

For the purposes of this problem, assume that the bullet has a constant velocity.

Locations will be input and output as Cartesian triples (x,y,z), where x, y, and z are integers between -1000 and 1000 (inclusive).

Input
The first line of input will contain a single integer n indicating the number of bullets.

The following n lines will each contain two Cartesian triples separated by a single space. The first triple represents the starting position of the bullet when the gun is fired. The second triple represents the position of the bullet one second after the gun is fired.

Output
Output the Cartesian triple representing the position of the bullet 5 seconds after the gun is fired.

Example Input File
```
3
(0,0,0) (1,1,1)
(0,0,0) (-3,27,0)
(-1000,-1000,1000) (-700,-1000,999)
```

Example Output To Screen
```
(5,5,5)
(-15,135,0)
(500,-1000,995)
```

Problem 5 **72 Points**

Tales From the Crypt

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

Some secret communications use a set of publicly-available text (such as the Declaration of Independence) as a key. If only the sender and recipient know the key, the sender can encode a message, character by character, by finding an instance of the same character in the key and using its position to create the encrypted message. Since the recipient already knows the key, he can take the encrypted message, which is just a list of character locations in the key, and reconstruct the original message.

Given a paragraph of text (the key) and an encrypted message, write a program that can display the decrypted message.

Input

The input to this problem will be split into two parts: the key, and a list of messages to decrypt.

The first line of input will contain a single integer *n* indicating the number of lines making up the key (less than 10). The following *n* lines contain the text that makes up the key. Each line will contain no more than 100 characters.

The next line of input will contain a single integer *m* indicating the number of messages to decrypt. The following *m* lines each contain a message that needs to be decrypted. Each messages is a non-empty series of up to 20 space-separated ordered pairs of the format "*x*,*y*". Each ordered pair corresponds to exactly one character in the key where *x* is the line and *y* is the column where the character occurs in the key. The line and column positions are unit indexed (i.e., they start at 1).

Output

For each encrypted message in the input, output the decrypted version on its own line.

Example Input File

```
6
When in the Course of human events, it becomes necessary for one people
to dissolve the political bands which have connected them with another,
and to assume among the powers of the earth, the separate and equal station
to which the Laws of Nature and of Nature's God entitle them, a decent
respect to the opinions of mankind requires that they should declare the
causes which impel them to the separation.
3
6,1 1,6 2,17 4,5 6,5 5,1
6,10 1,5 4,14 2,2 2,10 6,5 2,3 1,13 3,1 5,31 5,72 6,42
5,4 5,1 2,2 3,19 5,1 6,2 3,12 3,12 1,6 3,2 3,19 3,4 1,6 6,6 3,7 1,58 1,24 1,4
```

Example Output To Screen

```
cipher
i Love Cake.
programming is fun
```

Problem 6 | **72 Points**

What's Not On The Menu

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

Ebola is a contagious disease that cafeteria workers don't like. Since they can't control what kinds of contagious folks come in the door, they take every possible precaution to prevent the spread of disease with the notable exception of cleaning the food trays. They have only one tray stack, and customers both get trays from and return trays to the top of the stack without any cleaning.

Tracking an Ebola infection through a tray stack can be tough work. Each tray has two sides, each of which can be infected or not infected. People can infect trays, trays can infect people, and trays can infect each other. Here is a list of exactly what actions can transmit the virus:

- If an infected person takes a tray, that tray becomes infected on both sides.
- If any person takes a tray that is infected on either side, that person becomes infected and spreads the infection to both sides of the tray.
- When an infected tray is returned to the tray stack, it is placed on top of another tray, whose top side then becomes infected.
- If any tray is returned to the stack and placed on top of another tray whose top side is infected, the bottom of the new tray is infected.

Tracking the virus would be simplified if these rules could be put into a computer model, so it's your job to write a program that uses the above rules to determine which customers are infected by analyzing the order in which they take and return trays.

Input

The first line of input will contain a single integer *n* indicating the number of data sets to be processed.

Each data set will consist of two lines:

1. The first line will contain a single integer *m* (between 1 and 20, inclusive) indicating the number of 'tray actions' performed by customers.
2. The second line will contain a space-separated list of *m* tray actions. A tray action is a two-part string of at most 20 characters that identifies a person and indicates whether they take a tray from, or return a tray to, the tray stack. The first character in the string is a 'g' if the customer gets a tray or a 'r' if the customer returns a tray. The remainder of the string is the customer's name.

Please make the following assumptions:

At the beginning of each data set, all trays are in the tray stack and no trays are infected.
In a given data set, the first person performing a tray action is the only person infected before getting a tray.

Output

For each data set, output a single line containing the names of customers that are infected. List them in the order they become infected, and do not list the same customer multiple times.

Example Input File

```
3
1
gEbolaJoe
3
gBob rBob gJack
11
gBob gJack rBob gCarl gLenny gHomer gMarge rLenny rJack gBurns gSmithers
```

Example Output To Screen

```
EbolaJoe
Bob Jack
Bob Carl Lenny Burns Smithers
```

Problem 7 **72 Points**

Word Wrapper

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

"Word wrap" is a feature that causes a word processor to fit text within the margins of a document by putting as many words as possible on one line before automatically wrapping to the next line. In the event that a single word is bigger than the space allowed by the margins, the word will appear on its own line. Given an unformatted line of text and a number of columns, output the text so that it wraps at a given number of columns.

Ex:

Given the text:

```
Give a man a fish, and feed him for a day.
```

And a number of columns, 8, the resulting text would be:

```
Give a
man a
fish,
and feed
him for
a day.
```

Also, given the text:

```
Teach a man to fish, and feed him for a lifetime.
```

And a number of column, 8, the resulting text would be:

```
Teach a
man to
fish,
and feed
him for
a
lifetime.
```

Note:
- Words are kept as contiguous units. A word is defined as a contiguous string of non-whitespace characters (e.g., "fish," and "day." are words).
- Each line must be *x* columns or less, where *x* is the number of columns specified, unless the line contains a single word whose length is greater than *x* (e.g., the length of "lifetime." is greater than 8).
- Spaces between words on different lines are deleted when wrapping text.

Input

The first line of input will contain a single integer *n* indicating the number of data sets. Each data set will consist of two lines:
1. An integer *x*, where $1 <= x <= 50$, specifying the number of columns for the output text.
2. A line of text, which consists of a series of words, each separated by a single space.

Output

The output for each data set will be the input text, formatted within the specified number of columns as described above.

Example Input File

```
3
8
Give a man a fish, and feed him for a day.
12
A stitch in time saves nine, but a safety pin is quicker.
16
What goes around, comes around.
```

Example Output To Screen

```
Give a
man a
fish,
and feed
him for
a day.
A stitch in
time saves
nine, but a
safety pin
is quicker.
What goes
around, comes
around.
```

Problem 8		72 Points

Dumb Waiter

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

Working in a restaurant is never easy, but some waiters have worse luck than others. In an attempt to determine which waiter has the worst luck and will have the hardest time cleaning up broken dishes, write a program that can analyze a 'picture' of a set of broken dishes and determine the number of separate pieces.

Input

The first line of input will contain a single integer *n* indicating the number of data sets (sets of broken dishes). Each data set will consist of two parts:

1. The first part is a line containing a single positive integer *m* (less than 10) indicating the number of lines that will be use to describe the 'picture' of the broken dishes.
2. The second part is a series of *m* lines, each of length strictly less than 10 characters, and quite possibly empty. These lines will be filled with a combination of blanks and pound characters ('#') which represents the picture of the broken dishes on the floor as seen from above. Each piece is defined as a set of connected pound characters, where two pound characters are connected only when they are vertically or horizontally adjacent (i.e., diagonal adjacency does not imply a connection).

Output

For each data set, determine the number of distinct pieces pictured and output that integer on its own line. You can safely assume that there will be at least one piece in each picture.

Example Input File

```
3
4
  ###
  # #
 #
###
9

  #  ###
 #   ##
  #
     #
  ###
  ###
  #

2
### #
   ##
```

Example Output To Screen

```
2
6
2
```

Problem 9 | **72 Points**

Lather, Rinse, Repeat

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

You are a veterinarian technician for the local clinic. Actually, in your case, your job title is a fancy way of saying you shampoo dogs. But you take your dog-shampooing duties seriously, and decide to write a program to determine the minimum cost necessary to shampoo a given set of dogs. You know the following about the cost of shampoo:

8 oz. bottle of shampoo = $5
16 oz. bottle of shampoo = $9
24 oz. bottle of shampoo = $12

And you know the following about dogs:

Small dogs require .5 oz. of shampoo.
Medium dogs require 1 oz. of shampoo.
Large dogs require 2 oz. of shampoo.
Extra-large dogs require 4 oz. of shampoo.

Input

The first line of input will contain a single integer *n* indicating the number of data sets. Each data set will consist of a single line containing the number of small, medium, large, and extra-large dogs, respectively, each separated by a single space.

Output

The output for each data set will be the minimum cost of the shampoo necessary to shampoo the given set of dogs, in the format $*x*, where *x* is the cost in dollars. Note that bottles must be purchased in full amounts.

Example Input File

```
3
1 1 1 1
0 2 3 6
16 10 6 3
```

Example Output To Screen

```
$5
$17
$24
```

Problem 10 | **72 Points**

Amazing Mouse

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

Write a program that simulates the behavior of a mouse in a maze.

As viewed from overhead, the maze is made of discrete square units. Each unit is either a wall, an open space, an open space with a piece of cheese, or an open space containing the mouse (exactly one). The mouse can see across open space in the four cardinal directions but cannot see through walls.

If the mouse can see at least one piece of cheese, he will move one unit toward the piece closest to him. When the mouse moves and occupies the same square as a piece of cheese, he eats it, removing the cheese from the maze. If there is no closest piece of cheese (i.e., there are two or more pieces that tie for the closest or he can see no cheese) then the mouse can't decide what to do next and the simulation ends.

Input

The first line of input will contain a single integer *n* indicating the number of simulations to run.

Each simulation consists of the following input:

1. A single line containing a single integer *m* from 1 to 10 (inclusive) indicating the size of the maze (*m*x*m*)
2. The next *m* lines will each contain *m* characters and represents the maze. Possible characters are:
   'm' – The mouse. There will be exactly one mouse in each maze.
   '#' – A wall.
   '.' – An open space.
   'C' – A piece of cheese.

Note that the only assumption you can make about the maze is that there is exactly one mouse. It is possible to have a maze without walls, without cheese, or without open space.

Output

For each simulation, output a picture of the maze at the end of the simulation. Use the same symbols as used in the input. Do not print the dimensions of the maze.

Example Input File

```
3
5
#####
#m.C#
#.#.#
#..C#
#####
4
####
#mC#
#C.#
####
7
.......
.m....C
.#.#...
...C.C.
.......
.C....C
...C...
```

Example Output To Screen

```
#####
#...#
#.#.#
#..m#
#####
####
#mC#
#C.#
####
.......
.......
.#.#...
.......
.......
.C....C
...m...
```