University Interscholastic League

Computer Science Competition

2014 Invitational A 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. Table of Contents

1. Candy Fest

Program Name: CandyFest.java Input File: candyfest.dat

Your high school is celebrating its annual technology Candy Fest this year. The Chef has agreed to provide the candy for this fest. Her assistant has arranged N boxes of candy in linear order, but each box has a random number of pieces of candy. Since she is a perfectionist she needs to make sure all boxes have the same number of candies with as many as possible in each box.

Here are the requirements for this exercise.

- Her goal is to have the maximum possible number of candies in each box, with the same in each box, with a minimum of one candy per box.
- To minimize effort and reduce the amount of times the candies are touched, she will transfer candies only from left to right, NOT from right to left.
- She will start with the left most box and make only one move per box, again from left to right.
- If the box contents are OK, no move is necessary.
- She can also remove any number of candies from the last box and throw them out.

For example, given three boxes with 3 2 and 1 candies in each box, she can transfer 1 candy from the first box to the next box, which makes the configuration 2 3 1. Then she can transfer 1 candy from the 2nd box to the 3rd, leaving 2 candies in each box for a final configuration of 2 2 and 2, with no candies thrown out at the end.

In another example with 1 2 3 and 4 candies in four boxes, since the first box only has one candy, the rest of the boxes can only have one and you end up throwing out 6 candies out of the last box, leaving only 1 candy in each box.

Since you are really good at mathematics and logic, she has asked you to help her by writing a program to figure this out.

Input

The first line of input contains a single integer T, the number of test cases. Each test contains 2 lines. The first line contains a single integer N, the number of boxes. The next line contains N integers, indicating the number of candies in each box.

Output

For each row of boxes, output the number of candies each box will contain after transferring candies as described above.

Constraints:

No boxes will be empty initially, and there will be no more than 10 boxes with no more than 100 candies in any one box.

Example Input:

```
3
3
3 2 1
4
1 2 3 4
3
3 1 2
```

Example Output:

```
2
1
2
```

2. Ciao Mondo

Program Name: Ciao.java Input File: None

Say “Hello World” in Italian, French, and Spanish. Hello is “Ciao” in Italian, “Bonjour” in French, and “Hola” in Spanish. World is “Mondo” in Italian, “Monde” in French, and “Mundo” in Spanish.

Input
None.

Output
Print out the phrase in Italian, French and Spanish, one phrase per line. Keep the capitalization as shown above, and have one space between words, and no space or punctuation at the end.

No Input File

Example Output
```
Ciao Mondo
Bonjour Monde
Hola Mundo
```

3. Common Names

Program Name: CommonNames.java Input File: commonnames.dat

Have you noticed how first names go in and out of fashion? Ezekiel and Martha are hardly popular now, and in a generation, Brittney and Ashley will only be known among grandmothers. You are to determine the most popular name from a list we provide.

Input

The first line has the number of test cases, T. T < 100. The next few lines have N (N < 100) names separated by spaces. A name will only have alphabetic characters, without space, and will have only one uppercase letter -- the first one. A test case will always have at least one name.

Output

For each test case, output the most popular name. If more than one is equally popular, output all the popular ones in lexicographical order, separated by single spaces, one line for each test case. There should be no space at the end of a line.

Example Input

```
2
Martha Mary Jane Mary Mary Jane Jane
Johnathan
```

Example Output

```
Jane Mary
Johnathan
```

4. Connect Four

Program Name: ConnectFour.java Input File: connectfour.dat

You might have played the Connect Four game, where players alternately put red and blue tokens in at the top of a column chosen from an array of 7 columns 6 rows high. There are 21 tokens of each color, so it is possible to completely fill in the board. The players take turns placing tokens at the top of each column, which then fall to the bottom of the column, or on top of any token already there. A win is when a player has 4 of his own tokens in a row -- vertically, horizontally, or diagonally. Write a program to determine if the given pattern of 'r' and 'b' tokens is a win or not, and if so, who won the game.

Input

The first line is N, indicating the number of 6X7 game boards that follow.
Each character on the board is either an 'r', 'b' or '-', indicating a red token, blue token, or empty space. All boards indicate a real game, with blanks only at the tops of the columns (no magically suspended tokens).

Output

For each test case, output "Red Wins", "Blue Wins", or "No win".

Example Input

```
3
-------
-------
---r---
brbr---
brbr---
brbr---
-------
-------
-------
-------
rrr----
bbbb---
rbbrrrb
rbrbbbr
bbrbrrb
brrrbrr
brbbrrb
rbbrbbr
```

Example Output

```
Red Wins
Blue Wins
No win
```

5. Family

Program Name: Family.java Input File: family.dat

You have decided to create your family tree, but it has been very confusing. You have a lot of information about parent-child-sibling relationships within your family, but you are having trouble organizing it. Can you write a program to help?

Input

The first line is N, the number of relationship lines that follow. The next N lines are of the format *Name1 relationship Name2* where Name1 and Name2 are exactly one word each (only alphabetic characters) and relationship is father, mother, or sibling (i.e. brother or sister). It can be read as "Name1's <relationship> is Name2". A person will have at most one father and one mother listed. There will be no inconsistencies such as a person having two relationships with one person.

Output

For each person, output his or her name on a line. Then on the next line, put the word Father, followed by a colon and a space, followed by the person's father. If not in the given database, put Unknown. Then do the same for Mother, and Siblings. Each sibling should be listed on a separate line. Do not assume relationships that are not explicitly stated in the input; for example, even if two people are siblings, don't infer that they have the same father or mother. If a person's sibling is listed, assume that the relationship is mutual. However, do not assume that the sibling relationship is transitive i.e. if A is listed as B's sibling, and B is listed as C, do not automatically assume that A and C are siblings.

The names are alphabetically listed, and so are each person's siblings. If no siblings are known, don't output anything. Leave a blank line between two people's data.

Example Input

```
6
John father Jake
Mary sibling John
Nigel father Jake
Mary mother Amanda
Mary sibling Aaron
Mary sibling Zeke
```

Example Output

```
Aaron
Father: Unknown
Mother: Unknown
Sibling: Mary

Amanda
Father: Unknown
Mother: Unknown

Jake
Father: Unknown
Mother: Unknown

John
Father: Jake
Mother: Unknown
Sibling: Mary

Mary
Father: Unknown
Mother: Amanda
Sibling: Aaron
Sibling: John
Sibling: Zeke

Nigel
Father: Jake
Mother: Unknown

Zeke
Father: Unknown
Mother: Unknown
Sibling: Mary
```

6. Farkle

Program Name: Farkle.java Input File: farkle.dat

Farkle is a game where players throw dice, and based on the score seen, employ different strategies. In this program, you will throw a 6-sided fair die (with numbers 1-6 on the six faces) repeatedly until the same number turns up several times in a row; this is called a round. This round is repeated 100 times, and the average number of rolls before the round ends is calculated and reported. For example, if we say that the same number has to appear 3 times in a row, the first two rounds might look like below, where first 3 appears thrice and then 4 appears thrice.

```
1221333
4634125444
```

The average of the number of rolls for these two rounds is (7 + 10)/2 = 8.5. Note that in your actual program, you will have 100 rounds instead of 2.

You will use the Random class from java.util, with its nextInt method to simulate the die roll by calling rand.nextInt(int), where rand is a Random class object. One game consists of:

(1) creating a Random object with the provided seed by setting an object of class Random to Random(seed_value)
(2) calling nextInt repeatedly until the same number appears several times in a row
(3) repeating steps 1 and 2 100 times and
(4) reporting the average number of rolls per round (averaged over the 100 rounds), rounded to the nearest single decimal place.

Input

The input file has several line with two integers on each line. The first is the seed value to be used for the game. The second is a number between 2 and 5 inclusive that specifies the number of times a roll value must appear before each of the 100 rounds per game is complete.

Output

Print all the rolls of only the first round on a line, followed on a new line by the average number of rolls for one hundred rolls, rounded to the nearest single decimal place.

Example Input

```
123 3
1234 2
124 2
223 4
```

Example Output

```
333
38.2
366
7.7
355
6.7
261361542512132332363552413634632322624444
263.0
```

7. Game of Life

Program Name: GameOfLife.java Input File: gameoflife.dat

Conway's game of life tries to show how the functions of life -- birth, growth, reproduction and death -- can be mimicked by very simple rules. For this program you will be given a 2-D grid with 1's (representing living cells) and 0 (representing dead cells). This 2-D world evolves in time as per simple rules. The rules are the following:

1. If a cell is immediately surrounded by more than 3 creatures (counting up its neighbors in the 8 directions left-right, up-down, and diagonally), it dies by overcrowding.
2. If a cell has two or three neighbors, it lives for the next round.
3. If a cell has less than two neighbors, it dies due to lack of companionship.
4. A dead cell with exactly three neighbors comes alive in the next round, as if by reproduction.

The initial pattern constitutes the *seed* of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed—births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a *tick* (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

Input

The first N+2 lines have the data for each test case.
The first line is N, the size of the square grid.
The next line is M, the number of generations your program has to let the grid world evolve.
The next N lines are 1's and 0's representing each row's cells.

Output

Print the grid world as it would look after M generations. You may assume that any cells beyond the boundary are dead. A blank line separates multiple outputs.

Example Input

```
2
5
3
00110
01010
01110
00110
00000
10
5
0000111110
1010000000
1001011110
1111001100
0000001100
0000111000
0000011000
0000011100
0000000110
0000001110
```

Example Output

```
00110
11001
11011
01110
00000

0000000000
0000000000
1010000000
0010000000
1010000000
0000000000
0000000000
0000001000
0000001000
0000001000
```

8. Italian Flag

Program Name: Flag.java Input File: flag.dat

Now that we are on a European roll, for the next problem, print out the flag of Italy using the ASCII characters `G`, `' '` (space), `R`, and `'-'` (dash). The flag of Italy is a rectangle divided by vertical lines into green, white and red zones, equal in width to each other. The Green and Red zones will be filled by G and R, with no boundary characters. The white zone will be spaces, bounded on the top and bottom by the hyphen (dash) character.

Input
The first line of input will be N, the number of test cases. N < 10. Each test case will have two integers on one line. The first integer is the width, divisible by 3 and >= 6. The second is the height, >= 3.

Output
For each test case, draw the flag of Italy as specified. One blank lines needs to separate each output.

Example Input
```
2
12 5
15 3
```

Example Output
```
GGGG----RRRR
GGGG    RRRR
GGGG    RRRR
GGGG    RRRR
GGGG----RRRR

GGGGG-----RRRRR
GGGGG     RRRRR
GGGGG-----RRRRR
```

9. Obstacles

Program file: Obstacles.java **Input file: obstacles.dat**

You are taking part in a Robotics Competition. Robots have to negotiate a floor with obstacles for this competition. However, you could not finish making your robot before the given day, and your robot can be made to move only in two directions -- eastward and southward (right and down). Can you help figure out how many ways there are for the robot to go from entrance to exit?

The floor is laid out in the form of a rectangular grid, with `N` rows and `M` columns. The robot will start from the top left corner, (0,0), and exit at the bottom right (`N`-1, `M`-1). Positions that the robot can occupy are marked by '.' and obstacles are marked by '*'. The task is to find the number of paths from (0,0) to (`N` -1, `M`-1).

Input

The first line of input has the number of test cases `T`. For each test case, the first two lines will contain the integers `N` and `M` denoting the number of rows and columns in the floor respectively. This will be followed by `N` lines. Each line will have m characters – either '*' or '.' denoting an obstacle or clear path, respectively.

Output

Print the number of paths the robot can take to exit the grid.

Example Input:

```
3
2
2
.*
..
2
3
.*.
*..
5
4
....
.*..
.*..
....
***.
```

Example Output:

```
1
0
5
```

10. Roman Numerals

Program Name: Roman.java Input File: roman.dat

The object of this program is to convert from Roman numerals to Arabic. Roman numerals were used by the Romans until the current system came into widespread use.

In this system, numbers were represented by uppercase letters:

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

The value of a number written in Roman numerals is obtained by adding up the values of all the letters. Simple, right? The letters are by default written in decreasing order of value. There is one complication -- sometimes there are letter pairs where a lower valued letter is followed by higher valued one. The value of this combination is the difference of the values of the two in the pair. There will be no 'run' of three of more letters in increasing value order.

Input
The input file consists an initial value N, followed by N Roman numerals, one per line.

Output
For each Roman numeral, output that numeral followed by its Arabic (decimal) equivalent.

Example Input
```
3
XIX
MCMLIII
MMMIXIV
```

Example Output
```
XIX = 19
MCMLIII = 1953
MMMIXIV = 3013
```

11. Series

Program Name: Series.java Input File: series.dat

Find the Nth term of the given series whose ith term is a_i. The series is of the form:

$a_1 = A$

$a_n = B*C^n*a_{n-1}$

Input
The first line is T, the number of series definitions that follow. Each series definition contains 4 integers A, B, C, N separated by a single space.

Output
For each test case, output the Nth term.

Example Input
```
3
3 5 -1 5
1 2 1 4
-2 2 -2 3
```

Constraints
Absolute values of A and B are <= 5; -2 <= C <= 2; 2 <= M <=5

Example Output
```
1875
8
256
```

12. Triple Prime

Program Name: Triple.java Input File: triple.dat

Composite numbers that are the product of two large primes are used in cryptography, taking advantage of the fact that finding the factors of a large number is very difficult. In this problem, you will have to determine whether a number can be expressed as the product of not two, but three distinct primes.

Input

The first line is the number of test cases, T. T <= 20.
The next T lines each have an integer N, 2 <= N <=600.

Output

For each test case, output the number, followed by a list of the prime factors in ascending order (in the set format shown), and YES if the number can be expressed as the product of three distinct primes, or NO otherwise. The output format must be exactly as shown.

Example Input

```
4
8
30
105
210
```

Example Output

```
8 [2, 2, 2] NO
30 [2, 3, 5] YES
105 [3, 5, 7] YES
210 [2, 3, 5, 7] NO
```