PLU February 2023 Programming Contest

Novice Division

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. Problems will have either no input or will read input from a specified file. All output should be to standard output (the monitor).
3. All input is as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros.
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	Terms and Conditions
Problem 2	Tetral Downstack
Problem 3	Lines Sent
Problem 4	Spices
Problem 5	Blox
Problem 6	Revenge
Problem 7	Piece Bag
Problem 8	Perfect Clear
Problem 9	T-Spin
Problem 10	Scheduling Conflict
Problem 11	Shopping
Problem 12	Metrognome

1. Terms and Conditions

Input File: none

You are a contestant in a programming contest. Today, you're participating in an important contest. You've just arrived onsite, and the contest organizers need you print the Terms and Conditions of the contest.

Input
none

Output
Output the terms and conditions paragraph as shown below.

Example Input File
```
none
```

Example Output to Screen
```
Terms and Conditions of the Tournament
1. Each team member must be a Washington State student.
2. You must use the assigned team computer to solve problems.
3. Contestants may bring books and other printed materials.
```

2. Tetral Downstack

Input File: none

This is it. Everything you've ever trained for. The perfect opportunity. You've pressured your opponent while maintaining a clean stack yourself. All you need to do to finish them off is one more four-line clear, and – Oh no! You've dropped your I piece. Quick, draw yourself in another I piece so you can complete your downstack and win the game!

Input
none

Output
Output the piece as shown below.

Example Input File
```
None
```

Example Output to Screen
```
.-----.
|     |
|     |
|-----|
|     |
|     |
|-----|
|     |
|     |
|-----|
|     |
|     |
.-----.
```

3. Lines Sent

Input File: lines.dat

The match is over, and you've won! Unfortunately, the system seems to have malfunctioned slightly when trying to print the number of lines sent by each player. The tournament can't progress until this data is recorded, so you must figure out how many lines each player sent. Luckily, you know where this data is stored in the game's files, but you need to get it into a more presentable format. In your case, it just means adding some words to the end of the numbers.

Input

The first line will contain a single integer `n` that indicates the number of players that follow. Each data set will contain a single integer `x` on its own line denoting the number of lines that were sent by each player.

Output

For each player, output "`[x] lines were sent`", on its own line, where `[x]` is the number of lines sent.

Example Input File

```
6
50
45
70
35
44
60
```

Example Output to Screen

```
50 lines were sent
45 lines were sent
70 lines were sent
35 lines were sent
44 lines were sent
60 lines were sent
```

4. Spices

Input File: spices.dat

You have been cooking a lot, and some of your spices are very unorganized. You have decided to create a spice storage and catalog system. You must order the spices based on how often you use them, what color they are, how much you have, and then alphabetically.

Input

The first line will contain a single integer `n` that indicates the number of spices to follow.
Each of the following n lines will contain the spice name, and integer `m` denoting the amount of the spice, a rating of 1-5 denoting how often they are used (5 being the most, 1 the least), and the color of the spice.

Ordering Method

Order first by how often they are used, with 5 coming first and 1 last. Then order by color, with the preference chart for color being (colors appear as below, same capitalization) :

1) White
2) Red
3) Brown
4) Orange
5) Blue
6) Other (not the word other, just any other color)

Then order by how much you have, with the spices you have more of coming first.
Then order by name, alphabetically.
No two input lines will have the same values for all four items.

Output

Output the spice names in order based on the above ordering method.

Example Input File

```
4
Paprika 7 2 Red
Cumin 12 3 Brown
Sugar 8 3 White
Salt 3 5 White
```

Example Output to Screen

```
Salt
Sugar
Cumin
Paprika
```

5. Blox

Input File: blox.dat

You have started a new game called Blox recently, but given that you're too busy, you need to write a program to play for you. Blox is a game where if you're given blocks of certain heights, you try to see if you can stack them such that the stack is exactly a certain height.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will consist of a line of numbers, representing block heights, followed by a separate line with a number denoting the target.

Output

Output "`Blox be crazy`" if it is possible to form the height given with the given blocks, otherwise output "`Not on my Blox`".

Example Input File

```
3
2 3 4 5 6
7
1 2 3 4 4
12
7 7 7 7 7 7 7 7 7 7 7 7 7
8
```

Example Output to Screen

```
Blox be crazy
Blox be crazy
Not on my Blox
```

6. Revenge

Input File: revenge.dat

After being forced to take the punishment, you decide to get some revenge on the tournament organizers (even though you broke the rule first) in the most juvenile way: rearranging the letters on all the displays present. However, since you're not very good at coming up with funny phrases on the fly, you've decided to just move them systematically into an incomprehensible phrase. Your method works as follows: reverse the string, split the string in half, then repeat the process until each portion is only one character long. Then, combine all the portions back together to make one big scrambled phrase. If a string is of odd length, the second half is one character longer than the first.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will contain a phrase on a single line that may contain spaces.

Output

For each phrase, use the splitting and reversing method to rearrange the characters in the phrase, then output the result.

Example Input File

```
3
Concessions
Match Schedule
Restrooms down the hall and to the left
```

Example Output to Screen

```
issonnCosce
echledutMa Sch
dnt laal l tef oethoo mseRrsththe od wn
```

Explanation

For the first case, the process is as follows:

```
Concessions
snoissecnoC
snois secnoC
sions Conces
si ons Con ces
is sno noC sec
i s s no n oC s ec
i s s on n Co s ce
i s s o n n C o s c e
issonnCosce
```

7. Piece Bag

Input File: bag.dat

The classic game Tetris involves arranging falling tetrominoes on a board. There are seven different tetrominoes, each named after a letter that resembles their shape: J, L, S, Z, I, O, and T. In the original Tetris, the player would receive one tetromino at a time, and each tetromino would be chosen from among the seven possibilities independently and uniformly at random. This meant that any sequence of tetrominoes could appear in a game, such as numerous I tetrominoes in a row. Modern versions of Tetris remove these streaks by generating tetrominoes in groups or bags of seven: The first bag of seven tetrominoes in a game will be one of each of the seven different tetrominoes in a random order. The next bag of seven tetrominoes will also be one of each of the seven different tetrominoes in a random order (possibly but not necessarily different from the ordering of the first seven). Same goes for the next bag of seven, and so on and so forth. With this generator, it is still possible to get two of the same tetromino in a row (for example, the seventh and eighth tetrominoes in the game can be the same as each other), but it is not possible to get three of the same type in a row.

Given a sequence of tetrominoes, determine if the list represents a valid sequence of pieces. Note that the last bag may contain fewer than 7 pieces.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each of the following `n` lines will contain a sequence of letters representing the possible different shapes. These letters are `I`, `T`, `O`, `L`, `J`, `S`, and `Z`. The sequence will always start from the beginning of the bag.

Output

If the sequence of pieces has no duplicates within each bag (set of 7 pieces), output "`Valid bag`". Otherwise, output "`Invalid bag`".

Example Input File

```
3
SILTZJOJOLITSZZ
SILTZJOJOJITSZZ
Z
```

Example Output to Screen

```
Valid bag
Invalid bag
Valid bag
```

8. Perfect Clear

Input File: perfect.dat

Ah, the perfect clear. The easy yet elusive technique which captivates beginners and experts alike. The art of the perfect clear is one that is not easily perfected. That's why you've created a bot to help tell you if it's possible to perform a perfect clear under certain conditions! However, you're pretty low on funding, so you'll only be able to create the first step in telling if a board is perfectly clearable: the parity. Given a board, can you tell if there are an even number of squares or an odd number? The section of the board you will be provided is always 10 squares wide but can have any height.

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with a single integer `x` denoting the height of the board. The next `x` lines will have 10 characters each, which can be either `.` or `#`, where `.` represents an empty square and `#` represents a non-empty square.

Output

If there is an even number of non-empty squares, output "`Further investigation is needed.`". If there is an odd number, output "`Just give up already!`".

Example Input File

```
2
3
#..#.#...#
.........#
######.###
3
#..#.#.#.#
.........#
######.###
```

Example Output to Screen

```
Further investigation is needed.
Just give up already!
```

9. T-Spin

Input File: spin.dat

Lately, you’ve been having some trouble finding potential T-Spin setups on the board. So, you’ve decided to build yourself a training program to identify whether an area of the board has a potential T-Spin.

The section of the board is always 10 squares wide but can have any height. A T-Spin is possible if there are four empty squares in the shape of a T with one non-empty square blocking either the top left or top right of the T shape (but not both). Below are diagrams of the shapes you are looking for:

```
#..         ..#
...   or    ...
#.#         #.#
```

where `#` represents a non-empty square and `.` an empty square. If either of these 3x3 patterns are present on the board, a T-Spin is possible.

Input
The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will start with a single integer `x` denoting the height of the board. The next `x` lines will have 10 characters each, which can be either `.` or `#`.

Output
For each test case, if a T-Spin is possible, output "`T-Spin!!!`". Otherwise, output "`Missed it...`".

Example Input File
```
2
3
#..#.#...#
.........#
######.###
3
#..#.#.#.#
.........#
######.###
```

Example Output to Screen
```
T-Spin!!!
Missed it...
```

10. Scheduling Conflict

Input File: conflict.dat

With your success in the tournament, you decide to sign up in advance for the next one. However, you first need to find out if you will be busy on the next date. For some reason, the date of the next tournament is not displayed, but the signs instead say how many days it is until the next tournament. Given today's date, and the number of days until the next tournament, can you calculate the date of the next tournament?

Input

The first line will contain a single integer `n` that indicates the number of data sets that follow. Each data set will contain a date representing today in the format `mm/dd/yyyy`, as well as an integer `t` (1 <= t <= 1500) denoting the number of days between today and the next tournament.

Output

For each test case, output the date of the next tournament in `mm/dd/`yyyy format.

Example Input File

```
3
09/01/2021 20
01/01/2021 365
12/31/2021 1
```

Example Output to Screen

```
09/21/2021
01/01/2022
01/01/2022
```

11. Shopping

Input File: shopping.dat

Robby Goby is low on food and needs to go shopping for the things on his grocery list. He doesn't know curbside pickup is a thing and is going shopping the old-fashioned way, (he is indeed wearing a mask though). When he arrives, he realizes that the store is full and that he needs you to see if he can get everything on his list, in a preset order, through the crowd.

Input

The first line will contain a single integer `n` that indicates the number of stores (data sets).

Each data set will start with its dimensions and number of list items in `r,c,i` format, r being the number of rows, c being the number of columns, and i being the number of items on his list.

Each data set will have an `e` for the entrance/exit of the store, `#`'s for walls or people in your way, `.`'s for open paths, and letters that represent items.

There will be a list of those letters following the maze layout. Sometimes, the letters will also have names provided.

Output

Output `"It's grocery time!"` if it is possible to traverse the maze to reach everything on the list in order or `"It's not grocery time, sorry champ."` if not.

Example Input File

```
2
12,12,3
#####e######
#..........#
#.##..#..#.#
#.j#.#..#..#
#.##.c#.#..#
#.#...#.#..#
###...#.#.##
#..#..#.#..#
#.#..#..#..#
#p..#..#..##
#.##..#..#.#
############
j-jalapenos
c-carrots
p-potatoes
5,7,4
#######
#a...##
##.#d##
#b..#l#
##e####
a
l
b
d
```

Example Output to Screen

```
It's grocery time!
It's not grocery time, sorry champ.
```

12. Metrognome

Input File: metrognome.dat

Let me tell you, getting garden gnomes to dance in sync while listening to Moonlight Sonata is no easy task. The best we can hope for is that the garden gnomes eventually all line up on a single beat or two (since all garden gnomes dance at their own pace). More specifically, each garden gnome, G, has a certain value, Gi. At each integer multiple of Gi, (Gi, 2 * Gi, 3 * Gi, etc.), the garden gnome G will perform a dance move. Your task is to find out, for a given set of garden gnomes: what is the first point in time at which all the garden gnomes perform a dance move simultaneously?

Input

The input begins with the number t (1 <= t <= 100), the number of test cases to follow. For each of the following t test cases, a line with a single integer n (1 <= n <= 1000), the number of garden gnomes, will appear. The next line will consist of n space-separated integers G1, G2 … Gn (1 <= Gi <= 100).

Output

For each of the t test cases, print out a single integer: the first point in time during which all garden gnomes are dancing simultaneously. An answer is guaranteed to exist. Do not print trailing whitespace characters following and/or in between test cases.

Example Input File

```
3
5
1 2 3 4 5
2
1 2
3
7 8 3
```

Example Output to Screen

```
60
2
168
```