A. Smartness	Timelimit : 1000 MS	Memory Limit : 256 MB

While typing using the keyboard, we often make our attention to ensure that we are using the proper case. If the case is irrelevant, our writing loses its excellency and it is a bad habit to use inappropriate cases.

Now, check the following sample C code,

```
#include <stdio.h>

int main ()
{
   printf ("    cIty UNIversiTy PrograMMING club.    \n");

   return 0;
}
```

The problem of the output text is that there is no **SMARTNESS** in the text. The **SMARTNESS** of any text has the following conditions,

(i) There is no leading and trailing spaces in the text, but it may has spaces between the words.

(ii) Each first letter of a word must have the uppercase order. The rest of all the letters in a word must have the lowercase order.

Now, your job is to make the output text in such a way that we can call your output text too cool and **SMART** !!!

Sample Input	Sample Output
No input is available for this problem.	cIty UNIversiTy PrograMMING club.

B. Minimum XOR	Timelimit : 1000 MS	Memory Limit : 256 MB

Petya, Vashya and Rudra are very fond of binary numbers. One day they were playing an interesting game. In every round, each of them picked a 8-bit binary number, then they determined all the possible XOR (^) operations in binary representation between them and then calculated the minimum XOR value among them. But sometimes they felt that they had faced some troubles to calculate the exact result. So they made a phone call to BATMAN to come and write a program which can calculate the minimum XOR in binary among all the possible operations. BATMAN did his job perfectly. Now it is your turn. Are you ready?

Input Format: The input has only testcase which has three 8-bit binary numbers a,b

and c; each in a new line.

Output Format: The output will be the minimum XOR value in binary representation among all the XOR operations between a, b and c.

Sample Input	Sample Output
00000010 00000011 00000101 00000000 00111000 11111111	00000001 00111000

Clarification: You can safely assume that the decimal representation of the maximum binary input value does not exceed 255.

C. Help the Salesmen	Timelimit : 1000 MS	Memory Limit : 256 MB

City Mall is a very popular super-shop in Dhaka city. There are many good people who are engaged with that super-shop. The number of the customers of this super-shop is increasing day by day and now it becomes a severe problem for the stuffs. Sometimes when the crowd is too much, the whole environment becomes very noisy. It interrupts the salesmen to make the accurate calculation of total buying and selling system. As many people come here and buy their needs, the salesmen feel that if there will be a good monitoring system they will be able to handle the situation more precisely.

The authority called a meeting some days ago and now they would like to build a software which can solve their problem. By the rule of this software system,

(i) At most 10 people can enter into the super-shop at a time.
(ii) Each of them can buy at most 5 things he/she need.
(iii) Before buying the things each of the customer should choose a maximum fixed price (**MFP**) *which is less than or equal to the actual total price. If the actual total price exceeds* **MFP**, *he/she can not buy his/her needed things.*

After having these decisions, however, they come to know about you that you are a very good programmer. Now they wish to hire you to write the source code for their software. Can you help them out? ☺

Input Format: The input has only testcase which starts with an integer **n** (**1 ≤ n ≤ 10**) denoting the total number of the customers at a time. Then in each **n** lines there will be two integers, **P** (**1 ≤ P ≤ 1000**) and **t** (**1 ≤ t ≤ 5**), denoting the **MFP** of each customer should choose before buying and the amount of goods they want to buy respectively. Then in each next **t** lines there will be an integer $\mathbf{a_t}$ (**1 ≤** $\mathbf{a_t}$ **≤ 500**), denotes the price of the $\mathbf{t^{th}}$ good.

Output Format: For each customer, you have to print the actual total price **R** in the format "**R tk**". If this actual total price exceeds the given **MFP**, you should print a message "**Shomvob nah!**" without the double quotation.

Sample Input	Sample Output
2 50 2 40 25 1000 5 500 100 200 100 75	Shomvob nah! 975 tk

Explanation: In the sample input, there are two customers. Customer 1 chooses 50 tk total as the **MFP** but he tries to buy things which cost an amount of 40+25 = 65 tk total that exceeds 50 tk. On the other hand, Customer 2 chooses 1000 tk total as the **MFP** and he buys things which cost an amount of 500+100+200+100+75 = 975 tk total which is in the bound of 1000 tk.

D. Odven Function	Timelimit : 1000 MS	Memory Limit : 256 MB

From the divisibility rule of 2, we can classify the whole numbers into two types : Odd and Even. If the last digit of a number **N** be even, then **N** will be even; otherwise it will be odd. From the basic arithmetical properties we know that the $\mathbf{N^{th}}$ even natural number can be found from the formula **2N** and the $\mathbf{N^{th}}$ odd natural number can be found from the formula **2N-1.** Where **N = 1,2,3,.....**∞.

Now observing these stuffs, Mr. Newaj wants to coin a new function named *Odven function*. Given three integers x,y and k, Mr. Newaj defines the ***Odven function for x to y with respect to k*** by the following formula,

$$\Omega(x,y)_k = GCD\ (k^{th}\ even\ number \geq x,\ k^{th}\ odd\ number \leq y) \quad \text{(1)}$$

Here is the illustration of formula (1) , you are given a lower bound x and an upper bound y ($x \leq y$). Besides you are given another integer k. You first find the k^{th} even number **E** that is greater than or equal to x and the k^{th} odd number **O** that is less than or equal to y. Finally, you have to calculate the GCD of **E** and **O**. This is the desired value of Mr. Newaj's **Odven Function**.

Input Format: The input starts with an integer **T** (**T ≤ 10**) denoting the number of test cases. Then in each **T** lines there will be three integers **x**,**y** and **k** (**$1 \leq x,y \leq 10^6$**, **$1 \leq k \leq 5*10^5$**).

Output Format: For each testcase, you have to print the value of $\Omega(x,y)_k$ in the format, "**GCD a: (E,O) = b**", where **a** is the case number and **b** is the result. The explanation of **E** and **O** is already described above. It is ensured that the solution exists in every testcase.

Sample Input	Sample Output
2 2 10 2 100 1000 50	GCD 1: (4,7) = 1 GCD 2: (198,901) = 1

E. Modolas Jub	Timelimit : 1000 MS	Memory Limit : 256 MB

The orthography of the people of Modland is very interesting. For example, they write "Unposibol" instead of "Impossible", "Modolas" instead of "Modulus", "Jub" instead of "Job", "Caso" instead of "Case" and so on.

Specially, they like to perform modulo operation of two small numbers. They know that sometimes it is undefined to get the remainder of two numbers in some specific cases. They call this case "**Unposibol Caso**".

Now, your job is very simple. Given two integers **A** and **B**, your task is to determine whether we can perform a modulo operation by **A** and **B** (**A ≥ B)** or not. From the spelling system, we should call it, "**Modolas Jub**".

Input Format: The input starts with an integer **M** (**M ≤ 100**) denoting the number of total modulo operations. Then in each **M** lines there will be two integers, **A** and **B** (**0 ≤ A,B ≤ 10, A ≥ B)**.

Output Format: For each line of input, you should perform "**Modolas Jub**". If the result is undefined, simply print "**Unposibol Caso.**" without the double quotation; else you should print the result followed by a message "**Modolas Jub is successful.**" Print an extra new line after each testcase except the final one.

Sample Input	Sample Output
2 5 5 10 0	0 Modolas Jub is successful. Unposibol Caso.

F. Modland Calculator	Timelimit : 1000 MS	Memory Limit : 256 MB

We have already known from the very previous problem that the people of Modland have great enthusiasm about basic mathematical operations. When they get times, they make themselves busy with these actions. But now-a-days they has become too much lazy to make a basic calculation. So they need a primary calculator which can make all the basic mathematical works like addition, subtraction, multiplication, division and modulo operation.

Now they have some ideas about their primary calculator. Each time they open their

calculator, it can perform only one operation. Then it will be closed. After exact 2 seconds, it will be ready for its next calculation. That means it takes an interval of minimum 2 seconds to perform its next task. They have already set a prototype of their calculator display like the following,

Operand_1 *Operator* **Operand_2**

For example, if they give an expression like : **4 + 5**, the calculator takes 4 and 5 as the operands to sum them and finally gives the result **9**.

But astonishingly they believe that it is impossible to subtract, divide and mod a smaller number by the larger one. Besides, they do not support any modulo operation of negative numbers. The reason behind their such kind of believes has not been revealed yet.

Mendela, a small boy of Modland, starts to write a computer program that can function alike the calculator. But he is too noob to do it accurately and so he needs your help. Can you cooperate him?

Input Format: The input starts with an integer **T** (**T ≤ 100**) denoting the number of total tasks to be performed by the calculator. Then in each **T** lines there will be an integer operand **A**, an operator **X** and another integer operand **B** (**-1000 ≤ A,B ≤ 1000**).

Output Format: For each testcase, you should print the task number first and then the calculated result. For division, print the result upto two digits after the decimal point. If it is undefined or conventionally impossible to subtract, divide or mod; print a message "**Display Error.**"

Sample Input	Sample Output
5 5 + 5 67 - 70 45 * 2 2 / 0 -6 % 7	Task 1: 10 Task 2: Display Error Task 3: 90 Task 4: Display Error Task 5: Display Error

G. Expressions of Hatmitai | Timelimit : 1000 MS | Memory Limit : 256 MB

In ancient times, there was an isolated and deserted island in the pacific ocean named **Hirikiri**. The weather of Hirikiri is too much pretty. There were pleasant blooms all around the island. There was not too much hot or cold in the island. The roar of the ocean seemed a little bit scary but it sounded tremendous. One day, an unexpected incident happened. A young boy, **Gavinson** was shipwrecked off the coast of Hirikiri. He survived himself with extreme hardship. He managed to collect foods and drinks so that he can energize himself. But at one night, suddenly, Gavinson woke up from sleep and with his great surprise he saw an angel. The angle introduced himself as **Hatmitai**. Hatmitai attempted to generate some algebraic expressions. At last, he succeeded.

Hatmitai had exactly generated four expressions and then he declared that if anyone can solve these expressions with 100% perfection while given some values, Hatmitai would rescue that solver from this deserted island. Hearing this from far distance, Gavinson came forward and seeked Hatmitai to let him to give a try to solve all the expressions with some given values. The angel was very generous. He agreed and gave him the following expressions :

a. **$3*w^2$**
b. **$2*w + 4*x$**
c. **$8*w^2 + 5*x^3 + 10*y^4$**
d. **$10*w^5 + 4*x^2 + y + 9*z$**

The angel kept one variable in expression (a), two variables in expression (b), three variables in expression (c) and four variables in expression (d).

Gavinson was a surprise boy to do that calculation and he made Hatmitai pleased. So finally, however, he was rescued from the island with the help of Hatmitai.

Now, it is your turn to solve the above expressions with 100% perfection. Are you prepared enough?

Input Format: The input starts with an integer **T** (**T ≤ 800**) denoting the number of total lines. Then in each next **T** lines there will be,

a. **One w; or**
b. **One w and one x; or**
c. **One w, one x and one y; or**
d. **One w, one x, one y and one *z*; where 1 ≤ w,x,y,*z* ≤ 9**.

That means each line may contain one, two, three or four variables. You can safely assume that there will be no leading or trailing spaces in the lines.

Output Format: For each line, you should solve the expressions. The rule of solving the expressions is :

A. **If there is one value in the input line, you should solve expression (a).**
B. **If there is two values in the input line, you should solve expression (b).**
C. **If there is three values in the input line, you should solve expression (c).**
D. **If there is four values in the input line, you should solve expression (d).**

See sample I/O for clarification.

Sample Input	Sample Output
4 5 1 4 2 9 4 7 4 4 1	75 18 6237 168147

H. Sort???	Timelimit : 1000 MS	Memory Limit : 256 MB

John is in a new trouble now. His elder brother, Rimo gave him a little box of letters. The peculiarity of a letter is it can be not only a character of the Latin alphabet but also be a numerical digit between 0 and 9. If and only if all the letters letters from the box are found as a digit between 0 and 9, then John should sort the letters in descending order, or not.

Input Format: The input starts with an integer **T** (**T ≤ 20**) denoting the number of total testcases. Then in each next **T** lines there will be another integer **N** (**1 ≤ N ≤ 40**) denoting the number of the letters in a box. Then in the next line there will be **N** letters. Each letter can be a lowercase Latin character or a numerical digit.

Output Format: For each testcase, you should sort the letters and print them, if no Latin character is present in the box. Else print the message "**Trouble in position no. X!!!**", where **X** denotes the position of the character. If there are multiple Latin characters in the box consider the first appearance.

Sample Input	Sample Output
2 2 8 8 5 k m 5 9 0	8 8 Trouble in position no. 1!!!

I. RIP Bennington	Timelimit : 1000 MS	Memory Limit : 256 MB

You may have heard the name of Chester Charles Bennington. Haven't you?

Bennington was best known as the frontman for the rock band, **Linkin Park**. But he committed a suicide on July 20, 2017 at his own home in Los Angeles. All the devotees of Bennington got shocked at this premature and unexpected death. Richards is one of them. To give a tribute to his most favourite vocalist, he creates a playlist of the most popular songs of his band, Linkin Park. One day his little sister, Samitha, opens the playlist, enjoys some songs and then unfortunately makes a great mistake. A number of songs are deleted and now just ten songs exist randomly in the whole playlist.

a. Numb
b. In the End
c. Faint
d. What I've Done
e. Breaking the Habit
f. New Divide
g. Crawling
h. One Step Closer
i. Castle of Glass
j. Lost in the Echo

After coming home, Richards feels very sad and angry for this occurrence. He decides to punish Samitha. From any given line, Samitha has to find out if any name from the above list is hidden in the line or not.

Input Format: The input starts with an integer **T** (**T ≤ 20**) denoting the number of total testcases. Then in each next **T** lines there will be a line. The length of each line does not exceed 100.

Output Format: Print "**Bennignton still exists!!! :')**" if any name is found, otherwise print "**RIP Bennignton!!! :'(**", without the double quotations.

Sample Input	Sample Output
2 I wanna die in the ocean. But in the end it doesn't even matter.	RIP Bennignton!!! :'(Bennignton still exists!!! :')

J. Boltu in a hurry !!!	Timelimit : 2000 MS	Memory Limit : 256 MB

Boltu is a noob programmer. After solving some good problems in various online judges, he now moves to learn data structures. Today it is his 3rd day of data structures lesson. His teacher, Md. Amjad Ali wants to learn him **Queue** today.

In queue data structure, we call the 1st element **Front** and the last element **Rear**/**Back**. We have two basic operations, **Enqueue** and **Dequeue**. When we need to add any element to the back of the queue, we call this operation enqueue. When we remove any element from the front of the queue, we call this operation dequeue.

Boltu knows about these two operations from the class and then starts solving queue related problems. In his first problem, he is given some commands (such as, **ENQUEUE** and **DEQUEUE**) and he should perform all these commands very perfectly.

He should do it as soon as possible and he solves this just in 5 minutes. Can you be more faster ???

Input Format: Input starts with an integer **T (T ≤ 5**) denoting the number of total testcases. Each testcase starts with the number of total commands **N** (**N ≤ 20**). Then in each next **N** lines there will be three types of command.

ENQUEUE X
DEQUEUE
SUM

Here **X** (**-1000 ≤ X ≤ 1000**) denotes an element which should be added.

Output Format: Each testcase in a output starts with a line "**Case Z:**" where **Z** denotes the current testcase and then Print –

- "**X is added**" for **ENQUEUE**;

- "**Y is removed**" for **DEQUEUE,** where **Y** is the front of the queue;
- Sum of the front value and back value of the queue for **SUM**;
- "**Warning!!! Queue is empty**" when the queue is empty and dequeue is attempted.

See sample I/O for more clarification.

Sample Input	Sample Output
1 8 ENQUEUE 4 ENQUEUE -67 DEQUEUE ENQUEUE 1000 SUM DEQUEUE DEQUEUE DEQUEUE	Case 1: 4 is added -67 is added -67 is removed 1000 is added 1004 1000 is removed 4 is removed Warning!!! Queue is empty

Clarification: You can safely assume that DEQUEUE will be called at most one time after the queue being empty.