



International Collegiate Programming Contest 2026

Latin America Championship

March 6, 2026

Warmup Session

This problem set contains 4 problems; pages are numbered from 1 to 7.

General Information

Unless otherwise stated, the following conditions hold for all problems.

Program Name

1. Your solution must be called `codename.c`, `codename.cpp`, `codename.java`, `codename.kt`, `codename.py3`, where `codename` is the capital letter which identifies the problem.

Input

1. The input must be read from standard input.
2. The input consists of a single test case, which is described using a number of lines that depends on the problem. No extra data appear in the input.
3. When a line of data contains several values, they are separated by *single* spaces. No other spaces appear in the input. There are no empty lines.
4. The English alphabet is used. There are no letters with tildes, accents, diaereses or other diacritical marks (ñ, Ã, é, Ì, ô, Û, ç, etcetera).
5. Every line, including the last one, has the usual end-of-line mark.

Output

1. The output must be written to standard output.
2. The result of the test case must appear in the output using a number of lines that depends on the problem. No extra data should appear in the output.
3. When a line of results contains several values, they must be separated by *single* spaces. No other spaces should appear in the output. There should be no empty lines.
4. The English alphabet must be used. There should be no letters with tildes, accents, diaereses or other diacritical marks (ñ, Ã, é, Ì, ô, Û, ç, etcetera).
5. Every line, including the last one, must have the usual end-of-line mark.

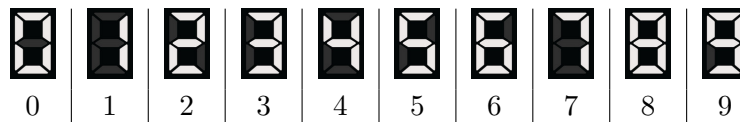
Problem A – LED Counter

Author: Alejandro Strejilevich de Loma, Argentina

A LED counter is a device having N positions arranged in a single row. Each position is able to display a digit from 0 to 9, and is made of seven LEDs, as this picture shows.



To display a specific digit, just the appropriate LEDs are turned on. The following picture indicates which LEDs are turned on for each possible digit.



As you can see, all but the middle LED are turned on to display the digit 0, the upper-right LED and the bottom-right LED are turned on to display the digit 1, and so on.

By turning on the appropriate LEDs in each of the N positions of the counter, the device is able to display 10^N distinct values, from $00 \dots 0$ (N zeros) to $99 \dots 9$ (N nines). Note that leading zeros are displayed.

Astro Void has owned a LED counter for quite some time, and so the device might have some malfunctioning LEDs. While a good LED turns on or off when needed, a malfunctioning LED is always turned on or always turned off, independently of the intended value of the counter.

Given the description of the state of each LED in Astro Void's LED counter (good LED that is turned on, good LED that is turned off, always-on LED, or always-off LED), you must tell the intended value of the counter, indicating which positions cannot be determined without ambiguity.

As an example of a LED counter with $N = 3$ positions, consider the picture below. If all the LEDs are good, then the intended value of the counter is of course 056. If the bottom LED of the first position is an always-on LED, then the intended value of the counter is still 056, because no other value would be displayed as the picture shows. However, if the bottom-left LED of the second position is an always-off LED, then the second position of the counter cannot be determined without ambiguity, since 056 and 066 would be displayed as shown.



Input

The first line contains an integer N ($1 \leq N \leq 10^5$) indicating the number of positions in the LED counter.

The i -th of the next N lines contains a string S_i of length 7 describing the seven LEDs in the i -th position of the counter. Each character of S_i describes a particular LED, from left to right and from top to bottom, that is, in the following order: upper-left, bottom-left, top, middle, bottom, upper-right and bottom-right. The character is either an uppercase letter "G" (good LED that is turned on), a lowercase letter "g" (good LED that is turned off), a plus sign "+" (always-on LED) or a hyphen "-" (always-off LED). It is guaranteed that S_i describes valid states for the LEDs in the i -th position of the counter. For instance, S_i is not "ggggggg", because when all LEDs are good, no digit would be displayed with all of them turned off.

Output

Output a single line with a string of N digits indicating the intended value of the counter. If a position of the counter cannot be determined without ambiguity, output the character “*” (asterisk) instead of the corresponding digit.

Sample Input 1

```
10
GGGgGGG
gggggGG
gGGGGg
ggGGGG
GggGgGG
GgGGGgG
GGGGGgG
ggGggGG
GGGGGG
GgGGGG
```

Sample Output 1

```
0123456789
```

Sample Input 2

```
3
GGGg+GG
GgGGGgG
GGGGGgG
```

Sample Output 2

```
056
```

Sample Input 3

```
3
GGGg+GG
G-GGGgG
GGGGGgG
```

Sample Output 3

```
0*6
```

Sample Input 4

```
2
+++gG--
---gG++
```

Sample Output 4

```
00
```

Sample Input 5

```
1
-+-+--
```

Sample Output 5

```
*
```

Problem B – Rent or Buy?

Author: Paulo Cezar Pereira Costa, Brasil

This problem is interactive.

Lola loves trying new things. Whenever she gets excited about a hobby, she goes all in.

Tennis? She buys the best racket she can find.

Skiing? She starts browsing for her own skis (even if the nearest slope is a thousand kilometers away).

Photography? Why not a professional camera?

The problem is that Lola's enthusiasm is unpredictable. Sometimes she practices a hobby many times. Sometimes she loses interest after just a few attempts.

After one too many expensive impulses, she realizes this is a common dilemma: should you keep renting each time, or invest upfront and hope you use it enough to make it worth it?

To solve this, Lola decides to build an app to automate this decision. For any given activity, a user can either:

- Rent the equipment for a cost of R per use.
- Buy the equipment for a one-time cost of B , after which each subsequent use costs a maintenance fee U .

The app must decide, for each new use, whether to continue renting or to finally make the purchase.

The difficulty is that the app does not know in advance how many times the activity will be done. The decision must be made one step at a time, without knowledge of future uses.

To keep things reasonable, the app must guarantee the following: For any possible number of times the activity is done, the total cost is at most twice the cost of the optimal strategy that knew this number beforehand.

Lola already built a beautiful interface for the app. Unfortunately, she lost interest when it was time to implement the actual decision-making logic.

That part is now up to you.

Interaction

First, read three integers R , B , and U ($0 \leq R, B, U \leq 10^6$), representing the cost of renting once, the one-time cost of buying, and the cost per use after buying.

After that, the interactor repeatedly sends either the input `use` or `end`.

For every input `use`, you must output exactly one of the following:

- **rent**: rent the equipment for this use (cost R),
- **buy**: buy the equipment now and use it immediately (cost $B + U$),
- **own**: use previously bought equipment (cost U).

You may output `buy` at most once. You may output `own` only if you have previously bought the equipment.

The interaction ends when you receive `end`. There will be at least one and at most 10^4 uses.

The interactor is **adaptive**: it may decide whether to continue or stop based on your previous outputs.

Read	Sample Interaction 1	Write
5 5 2 use		
		rent
use		
		buy
use		
		own
end		

Explanation of Sample 1:

Knowing that the the number of uses is 3, the optimal strategy is to buy at the first use, for a total cost of $B + 3U = 11$.

The solution that rents for the first use and then buys on the second use has a total cost of $R + B + 2U = 14$, which is less than twice the optimal cost [$2(B + 3U) = 22$].

Read	Sample Interaction 2	Write
5 100 0 use		
		rent
use		
		rent
end		

Explanation of Sample 2:

Even though there are no maintenance costs after buying, the cost of buying is so high that it's not worth it with only 2 uses.

Read	Sample Interaction 3	Write
5 5 0 use		
		buy
use		
		own
use		
		own
end		

Problem C – Coatless in Yakutsk

Author: Giovanna Kobus Conrado, Brasil

It was your first time visiting Salvador, and you made the rookie mistake of sleeping on the beach. You woke up red, sunburned, and frankly, humiliated. Swearing vengeance against the sun and all its terrible consequences, you decided that your next vacation would be somewhere with a real winter – like Yakutsk, Russia, where the average temperature is -42° Celsius.

But you came prepared! You brought a warm, cozy coat. The coat is perfect, it warms you very well. Maybe too well, as you get sweaty and the coat gets dirty after C days of use. Since your trip lasts more than C days, you must find a way to avoid walking around smelling bad.

To do so, when the coat gets dirty, you cannot wear it until it is washed, but you may also choose to wash it earlier. On any day you do not wear the coat – whether because it is dirty or being washed – you must endure the day's temperature without its protection. After being washed, the coat is fresh and ready to be worn again. At the start of your trip, the coat is clean.

Given the daily temperatures in Yakutsk for the duration of your trip, you must determine the lowest temperature on a day when you are forced to be without your coat, assuming you schedule wash days optimally to make this temperature as high as possible.

Input

The first line contains two integers C and N ($1 \leq C < N \leq 10^5$), indicating respectively the number of days you can wear the coat before it gets dirty, and the duration in days of your holidays.

The second line contains N integers T_1, T_2, \dots, T_N ($-50 \leq T_i \leq 50$ for $i = 1, 2, \dots, N$), where T_i is the temperature on the i -th day.

Output

Output a single line with an integer indicating the minimum temperature you must endure without your coat.

Sample Input 1

```
2 6
-20 -10 -5 -10 -2 -40
```

Sample Output 1

```
-5
```

Problem D – Game of Pieces

Author: Rafael Grandsire, Brasil

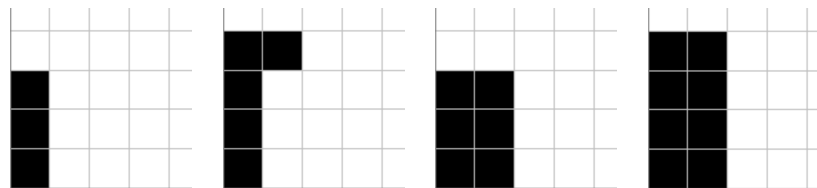
If you've ever played Tetris for long enough, you might have experienced the *Tetris effect* – seeing falling blocks even after you have stopped playing. To stay focused on solving problems and avoid such distractions, we will consider a simplified version of the game.

The game is played on a board composed of square cells arranged in a grid. Columns of the grid are numbered sequentially from left to right. The board is infinite to the right and to the top. Each cell is either empty or filled, and initially, all cells are empty.

A sequence of N rectangular pieces is given, and the pieces are dropped onto the board one at a time. Pieces have different sizes. A piece of size L is either vertical ($L \times 1$ cells) or horizontal ($1 \times L$ cells). When a piece is dropped at a specified column, it starts at a location above all the currently filled cells and falls straight down until it either reaches the bottom of the board or lands on top of an already filled cell. Once a piece lands, it fills its final set of cells.

Each time a piece lands, the game is considered *safe* if no empty cell has a filled cell above it; otherwise the game is *unsafe*, the offending piece is removed from the board, and the game continues with the next piece as if it had never been dropped.

In the example below, corresponding to the first sample input, the game becomes unsafe once the second piece lands, hence the piece is removed and the next pieces keep the game safe.



Given the sequence of pieces and their drop positions, your task is to determine, for each piece, whether it makes the board unsafe after landing.

Input

The first line contains an integer N ($1 \leq N \leq 2 \cdot 10^5$), indicating the number of pieces.

Each of the next N lines describes a piece with a character C and two integers L and P ($1 \leq L \leq 10^9$ and $1 \leq P \leq 10^{18}$), representing respectively the type of the piece, its length, and the position where it is dropped. For a vertical piece, C is the character “|” (pipe), the piece has $L \times 1$ cells, and it is dropped at column P . For a horizontal piece, C is the character “-” (hyphen), the piece has $1 \times L$ cells, and it is dropped spanning columns $P, P + 1, \dots, P + L - 1$.

Output

Output a single line with a string of length N . The i -th character of the string must be the uppercase letter “U” if the game becomes unsafe immediately after the i -th piece lands on the board, and the uppercase letter “S” otherwise.

Sample Input 1

```
4
| 3 1
- 2 1
| 3 2
- 2 1
```

Sample Output 1

```
SUSS
```

Sample Input 2

```
4
| 3 1
| 2 3
| 1 2
- 2 2
```

Sample Output 2

```
SSSU
```