# D. Census (census)

Time limit: 1 seconds
Memory limit: 128 MiB

A lesser-known fact about Cesenatico is that it is home to a secret society of $N$ female informaticians. This society is very secret indeed; no member knows any other. Each member has a unique ID: a non-negative integer $I$.

The only communication among members is indirect, via numbers scribbled with chalk at different locations across town. Every 100 years, the society performs a census to count its members. After the census is completed, every member should know the total number of members in the society.

The census takes place over multiple days. Each day, every member who is still taking part in the process will choose and perform exactly one action: to **read**, to **write**, or to **stop** taking part in it.

- If a member chooses to **read**, she picks a location $P$. During the daytime, she visits location $P$ and reads the number that is written there.
- If a member chooses to **write**, she picks a location $P$ and a number $V$. In the late evening, she visits location $P$ and changes the number that was written there to $V$. Since it is already dark, she cannot read the old number before she writes the new one.
- If a member chooses to **stop**, she no longer takes any actions in the following days.

If one member sees another write a number, she could recognize her. Therefore, it is strictly forbidden for two or more members to choose to write at the same location on the same day. (There is no such restriction for reading, as that can be done discreetly.)

If one or more members read from a location where another member wants to write on the same day, all reads occur before the write.

How should the society plan its census process in order to minimize the number of days until everyone learns the correct member count?

## Implementation

⇨ This is an interactive problem, in which an unknown number of instances ($1 \leq N \leq 100$) of your program will be executed simultaneously. Each instance simulates one member of the society.

There are $10^{18}$ locations. The number $P$ of a location must satisfy $0 \leq P < 10^{18}$. Initially, the value written at all locations is $V = 0$.

The new value $V$ written at a location must always be an integer such that $0 \leq V \leq 10^9$. In most subtasks, $V$ can only be 0 or 1. See the Scoring section for more details.

When an instance of your program starts, it should first read a line with two integers, $I$ and $M$ ($0 \leq I \leq M - 1$): the unique ID of the member of society represented by this instance and the total number of possible IDs. Within each test case, all instances will get the same value $M$ and distinct values $I$. Note that there may be IDs that are not assigned to any member.

Then, for each day in the census process, your program should choose the action it wants to perform and print a line accordingly:

| Action | Meaning |
|---|---|
| `r` $P$ | **Read** location $P$.<br>After printing this line, your program should read a line with the current value written at $P$. |
| `w` $P$ $V$ | **Write** at location $P$ the new value $V$.<br>If multiple instances write at the same $P$ on the same day, you will get the verdict *Not correct*.<br>Except for the examples and subtask 3, you must write $0 \leq V \leq 1$; see the Scoring section. |
| `!` $N$ | **Answer and stop**: report that there are $N$ members and stop taking part in the census. After answering, **your program should exit normally**. (Note that other instances of your program may continue running for additional days before they answer and exit.) |

If any instance of your program answers the wrong value of $N$, violates the protocol, uses more than 500 days, or exceeds the (per-process) time/memory limit, your submission will be judged as *Not correct* for the given test case.

Otherwise, your program will be *(Partially) Correct* on the test case and scored based on the value $D$: the maximum number of days any instance took to answer. For full score, you need to solve every test case with $D \leq 61$ and $V \leq 1$. See the Scoring section for details.

**Flushing.** If you are not using the provided templates, make sure to flush standard output after printing each line, or else your program might get judged as *Not correct*. In Python, this happens automatically if you use `input()` to read lines. In C++, `cout << endl;` flushes in addition to printing a newline; if using printf, use `fflush(stdout)`.

## Constraints

- $1 \leq N \leq 100$.
- $1 \leq M \leq 100\,000$.
- You may use at most 500 days.

## Scoring

Your program will be tested on several test cases grouped into subtasks. To obtain the score for a subtask, you must correctly solve all the tests it contains.

- **Subtask 0 [ 0 points]**: Examples (you can write any integer $0 \leq V \leq 1\,000\,000\,000$).
- **Subtask 1 [11 points]**: $M \leq 100$, and the $N$ members have IDs $0, 1, ..., N-1$.
- **Subtask 2 [12 points]**: $1 \leq N \leq 2$.
- **Subtask 3 [22 points]**: $M \leq 8000$, and you can write any integer $0 \leq V \leq 1\,000\,000\,000$.
- **Subtask 4 [55 points]**: No additional constraints.

**In subtasks 1, 2, and 4, you can only write $V = 0$ or $V = 1$ in each Write action.**

Let $X_s$ be the maximum points for subtask $s$ (shown above), and $D_s$ the largest number of days any of your programs uses on a test in subtask $s$. Then:

$$\text{score}_s = \begin{cases} X_s & \text{if } D_s \leq 61 \\ X_s \cdot \left(0.2 + 0.8 \cdot 1.01^{(60-D_s)}\right) & \text{if } 61 < D_s \leq 500 \\ 0 & \text{if } 500 < D_s. \end{cases}$$

The value of $\text{score}_s$ is rounded to the nearest integer per subtask, and your total score is the sum of these. To get the full score for the task, you need $D \leq 61$ and $V \leq 1$ on every test case.

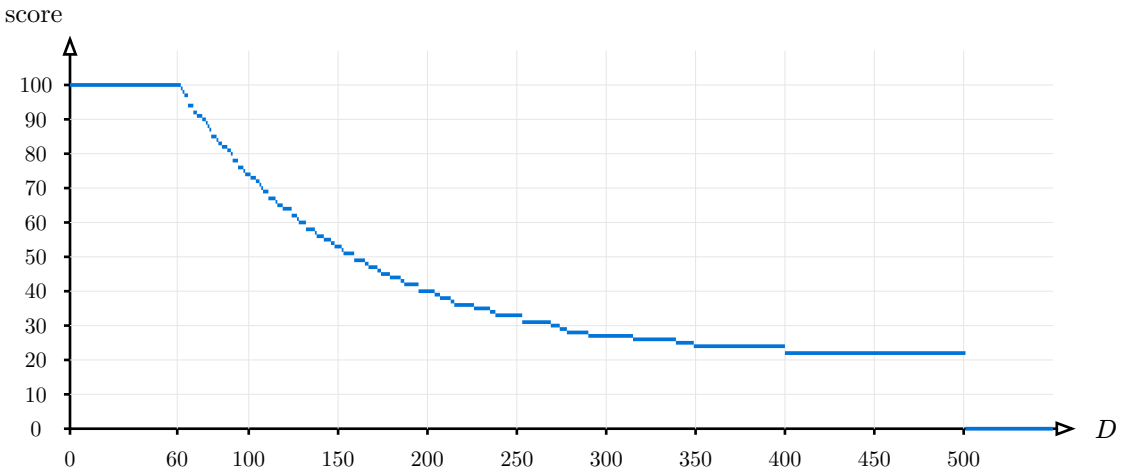


Figure 1: Total score, assuming every subtask is solved with the same maximum $D$.

## Examples

First example. Each pair of columns shows the communication between the grader and one instance.

| Gra. | Inst. 0 | Gra. | Inst. 1 | Gra. | Inst. 2 | Gra. | Inst. 3 | Gra. | Inst. 4 |
|---|---|---|---|---|---|---|---|---|---|
| 0 100 | | 1 100 | | 2 100 | | 3 100 | | 4 100 | |
| | w 12 1 | | w 50 1 | | w 99 0 | | w 7 1 | | r 5 |
| | | | | | | | | 0 | |
| | r 50 | | r 7 | | r 12 | | w 1 1 | | ! 5 |
| 1 | | 1 | | 1 | | | | | |
| | ! 5 | | r 1 | | w 0 0 | | ! 5 | | |
| | | 1 | | | | | | | |
| | | | ! 5 | | ! 5 | | | | |

Second example.

| Grader | Instance 0 |
|---|---|
| 0 8000 | |
| | w 0 0 |
| | w 1 1 |
| | r 2 |
| 1 | |
| | ! 2 |

| Grader | Instance 1 |
|---|---|
| 3 8000 | |
| | w 2 1 |
| | r 1 |
| 0 | |
| | r 2 |
| 1 | |
| | r 1 |
| 1 | |
| | ! 2 |

## Explanation

**First Example.** We have $N = 5$ members with consecutive IDs $0, 1, 2, 3, 4$ and $M = 100$ (valid for subtasks 1, 3, and 4). Instance $i$ corresponds to the member with ID $i$. The interaction above is just one possible legal sequence of operations and is **not** meant to be an efficient or sensible strategy; it is shown only to illustrate how the protocol works.

**Second Example.** We have $N = 2$ members, with IDs 0 and 3, and $M = 8000$ (valid for subtasks 2, 3, and 4). On the first day, the member with ID 0 writes a 0 at location 0 (no change), and the member with ID 3 writes a 1 at location 2.

| **location** | 0 | 1 | 2 | 3 | 4 | ... |
|---|---|---|---|---|---|---|
| **value** | 0 | 0 | 1 | 0 | 0 | ... |

On the second day, ID 0 writes a 1 at location 1, and ID 3 reads that same location. Note that the read happens during daytime, before the write in the evening. Hence, ID 3 still sees a 0.

| **location** | 0 | 1 | 2 | 3 | 4 | ... |
|---|---|---|---|---|---|---|
| **value** | 0 | 1 | 1 | 0 | 0 | ... |

On the third day, they both read location 2, where a 1 is written.

On the fourth day, ID 0 answers that there are 2 members (correct), while ID 3 reads the 1 at location 1. ID 0 immediately exits after this and does not participate in the coming days.

Finally, on day $D = 5$, the remaining member also correctly answers $N = 2$.

## Testing

To facilitate the testing of your solution, we provide a simple tool that you can download from CMS. The tool is optional to use. Note that the official grader on CMS is different from the testing tool.

To use the tool you need an input file. You can use the provided example inputs `census.input0.txt` and `census.input1.txt`, or make your own. The input file should start with the number of members $N$ and possible IDs $M$, followed by a line with $N$ numbers specifying the IDs of the society members.

For Python programs, say `census.py` (normally run as `pypy3 census.py`) run the testing tool as follows:

```
python3 testing_tool.py pypy3 census.py < census.input0.txt
```

For C++ programs, first compile your solution:

```
g++ -DEVAL -std=gnu++20 -O2 -pipe -static -s -o census census.cpp
```

and then run the testing tool:

```
python3 testing_tool.py ./census < census.input0.txt
```

Note that in this problem standard output is used for communication with the grader, so it should not be used for debugging. Instead, you may use the standard error output (stderr). In C++ you can use `cerr << msg << endl;`. In Python you can use `print(msg, file=sys.stderr)`.

The testing tool will read and present these stderr messages together with the queries performed by all your program instances. Note that for technical reasons they may show up slightly out of sync with each other.