# Problem A. Arrangement of Contest

| | |
|---|---|
| Input file: | `arrange.in` |
| Output file: | `arrange.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Little Dmitry and little Petr want to arrange a contest. Their little friends submitted several task proposals and now Dmitry and Petr want to select some of them for the contest. As they are just little boys, they cannot estimate quality of tasks, but they know for sure that in *good* contest title of the first problem starts with A, the title of the second one — with B, and so on.

Given titles of the proposed tasks, help little brothers to determine the maximal number of problems in a *good* contest they can arrange.

## Input

The first line contains single integer $n$ — the number of problem proposals received by the little brothers ($1 \le n \le 100$).

Next $n$ lines contain titles of proposed problems, one per line. The length of each title does not exceed 30 characters. Each title starts with an uppercase letter and contains only English letters, digits and underscores.

## Output

Output a single number — the maximal number of problems in a *good* contest. In case there is no *good* contest that may be arranged, output 0.

## Examples

| arrange.in | arrange.out |
|---|---|
| 12<br>Arrangement_of_Contest<br>Ballot_Analyzing_Device<br>Correcting_Curiosity<br>Dwarf_Tower<br>Energy_Tycoon<br>Flight_Boarding_Optimization<br>Garage<br>Heavy_Chain_Clusterization<br>Intellectual_Property<br>J<br>Kids_in_a_Friendly_Class<br>Lonely_Mountain | 12 |
| 3<br>Snow_White_and_the_7_Dwarfs<br>A_Problem<br>Another_Problem | 1 |
| 2<br>Good_Problem<br>Better_Problem | 0 |

# Problem B. Auxiliary Question of the Universe

| | |
|---|---|
| Input file: | `auxiliary.in` |
| Output file: | `auxiliary.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

As you probably know, scientists already discovered the Ultimate question of life, the Universe, and everything, and it is "What do you get if you multiply six by nine?". Not satisfied by this, the scientists contracted a small Magrateyan company to construct a mini-computer to find out some more specific question (they named it *auxiliary*), which can theoretically shed more light on life, the Universe or something else.

This computer was built, but unluckily (although not unexpectedly) the result of computation was corrupted and partially lost. Finally the computer constructors managed to receive a string, which is a part of the correct question. After thorough analysis the constructors started to believe that the original result can be reconstructed from the string by adding some letters to it without the string letters being reordered or removed. Also they believe that the correct result is an arithmetic expression (as with the Ultimate question), but since the question is auxiliary, it contains no multiplication, only addition. More precisely, it should correspond to the following grammar:

$$\begin{aligned} \langle \text{expression} \rangle &::= \langle \text{term} \rangle \mid \langle \text{term} \rangle \text{ `+' } \langle \text{expression} \rangle \\ \langle \text{term} \rangle &::= \langle \text{number} \rangle \mid \text{`(' } \langle \text{expression} \rangle \text{ `)'} \\ \langle \text{number} \rangle &::= \text{`0'} \ldots \text{`9'} \ [\langle \text{number} \rangle] \end{aligned}$$

The constructors do not want to risk again, and they need your help to give just something to their clients. They ask you to reconstruct the question based on the corrupted computer answer which they managed to retrieve.

## Input

The input file contains exactly one line — the corrupted auxiliary question. It is a non-empty string which is at most 1000 symbols long. This string contains only symbols '+', '(', ')', and '0', ..., '9'.

## Output

Output the reconstructed auxiliary question. It's guaranteed that there exists a correct question of less than 5000 symbols and your solution must also be shorter than that. If there is more than one solution, output any one.

## Example

| auxiliary.in | auxiliary.out |
|---|---|
| 1+0+1) | (1+0+1) |
| 2009 | 2009 |
| )(()( | (0)+((0)+(0)) |

# Problem C. Database

| | |
|---|---|
| Input file: | `database.in` |
| Output file: | `database.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Peter studies the theory of relational databases. Table in the relational database consists of values that are arranged in rows and columns.

There are different *normal forms* that database may adhere to. Normal forms are designed to minimize the redundancy of data in the database. For example, a database table for a library might have a row for each book and columns for book name, book author, and author's email.

If the same author wrote several books, then this representation is clearly redundant. To formally define this kind of redundancy Peter has introduced his own normal form. A table is in Peter's Normal Form (PNF) if and only if there is no pair of rows and a pair of columns such that the values in the corresponding columns are the same for both rows.

| | | |
|---|---|---|
| How to compete in ACM ICPC | Peter | peter@neerc.ifmo.ru |
| How to win ACM ICPC | Michael | michael@neerc.ifmo.ru |
| Notes from ACM ICPC champion | Michael | michael@neerc.ifmo.ru |

The above table is clearly not in PNF, since values for 2rd and 3rd columns repeat in 2nd and 3rd rows. However, if we introduce unique author identifier and split this table into two tables — one containing book name and author id, and the other containing book id, author name, and author email, then both resulting tables will be in PNF.

| | |
|---|---|
| How to compete in ACM ICPC | 1 |
| How to win ACM ICPC | 2 |
| Notes from ACM ICPC champion | 2 |

| | | |
|---|---|---|
| 1 | Peter | peter@neerc.ifmo.ru |
| 2 | Michael | michael@neerc.ifmo.ru |

Given a table your task is to figure out whether it is in PNF or not.

## Input

The first line of the input file contains two integer numbers $n$ and $m$ ($1 \le n \le 10\,000$, $1 \le m \le 10$), the number of rows and columns in the table. The following $n$ lines contain table rows. Each row has $m$ column values separated by commas. Column values consist of ASCII characters from space (ASCII code 32) to tilde (ASCII code 126) with the exception of comma (ASCII code 44). Values are not empty and have no leading and trailing spaces. Each row has at most 80 characters (including separating commas).

## Output

If the table is in PNF write to the output file a single word "`YES`" (without quotes). If the table is not in PNF, then write three lines. On the first line write a single word "`NO`" (without quotes). On the second line write two integer row numbers $r_1$ and $r_2$ ($1 \le r_1, r_2 \le n$, $r_1 \ne r_2$), on the third line write two integer column numbers $c_1$ and $c_2$ ($1 \le c_1, c_2 \le m$, $c_1 \ne c_2$), so that values in columns $c_1$ and $c_2$ are the same in rows $r_1$ and $r_2$.

## Example

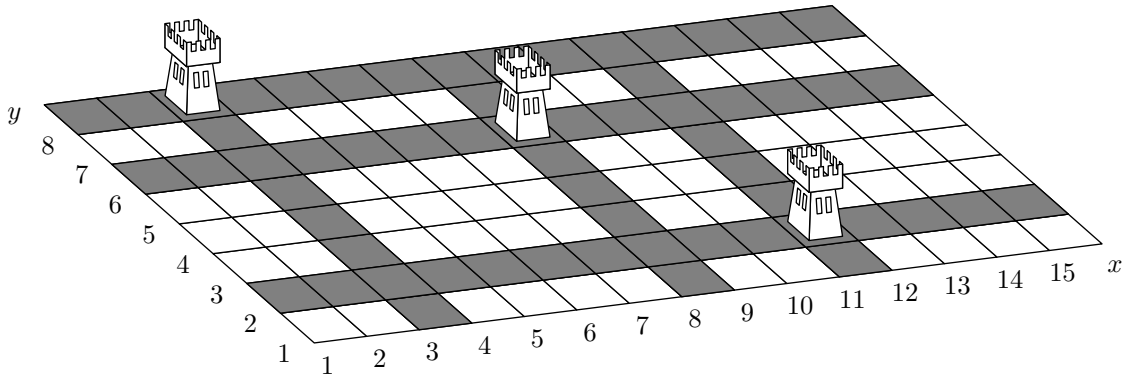
| database.in | database.out |
|---|---|
| 3 3<br>How to compete in ACM ICPC,Peter,peter@neerc.ifmo.ru<br>How to win ACM ICPC,Michael,michael@neerc.ifmo.ru<br>Notes from ACM ICPC champion,Michael,michael@neerc.ifmo.ru | NO<br>2 3<br>2 3 |
| 2 3<br>1,Peter,peter@neerc.ifmo.ru<br>2,Michael,michael@neerc.ifmo.ru | YES |

# Problem D. Defense of a Kingdom

| | |
|---|---|
| Input file: | `defense.in` |
| Output file: | `defense.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Theodore implements a new strategy game "Defense of a Kingdom". On each level player defends the Kingdom that is represented by a rectangular grid of cells. The player builds crossbow towers in some cells of the grid. The tower defends all the cells in the same row and the same column. No two towers share a row or a column.

The penalty of the position is a number of cells in the largest undefended rectangle. For example, the position shown on the picture has penalty 12.




Help Theodore write a program that calculates the penalty of the given position.

## Input

The first line of the input file contains three integer numbers: $w$ — width of the grid, $h$ — height of the grid and $n$ — number of crossbow towers ($1 \le w, h \le 40\,000$; $0 \le n \le \min(w, h)$).

Each of the following $n$ lines of the input file contains two integer numbers $x_i$ and $y_i$ — the coordinates of the cell occupied by a tower ($1 \le x_i \le w$; $1 \le y_i \le h$).

## Output

Output a single integer number — the number of cells in the largest rectangle that is not defended by the towers.

## Example

| defense.in | defense.out |
|---|---|
| 15 8 3<br>3 8<br>11 2<br>8 6 | 12 |

# Problem E. Explicit Formula

| | |
|---|---|
| Input file: | `explicit.in` |
| Output file: | `explicit.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider 10 Boolean variables $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$, and $x_{10}$. Consider all pairs and triplets of distinct variables among these ten. (There are 45 pairs and 120 triplets.) Count the number of pairs and triplets that contain at least one variable equal to 1. Set $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = 1$ if this number is odd and $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = 0$ if this number is even.

Here's an explicit formula that represents the function $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$ correctly:

$$f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = (x_1 \vee x_2) \oplus (x_1 \vee x_3) \oplus (x_1 \vee x_4) \oplus (x_1 \vee x_5) \oplus (x_1 \vee x_6) \oplus (x_1 \vee x_7) \oplus (x_1 \vee x_8) \oplus (x_1 \vee x_9) \oplus (x_1 \vee x_{10}) \oplus (x_2 \vee x_3) \oplus (x_2 \vee x_4) \oplus (x_2 \vee x_5) \oplus (x_2 \vee x_6) \oplus (x_2 \vee x_7) \oplus (x_2 \vee x_8) \oplus (x_2 \vee x_9) \oplus (x_2 \vee x_{10}) \oplus (x_3 \vee x_4) \oplus (x_3 \vee x_5) \oplus (x_3 \vee x_6) \oplus (x_3 \vee x_7) \oplus (x_3 \vee x_8) \oplus (x_3 \vee x_9) \oplus (x_3 \vee x_{10}) \oplus (x_4 \vee x_5) \oplus (x_4 \vee x_6) \oplus (x_4 \vee x_7) \oplus (x_4 \vee x_8) \oplus (x_4 \vee x_9) \oplus (x_4 \vee x_{10}) \oplus (x_5 \vee x_6) \oplus (x_5 \vee x_7) \oplus (x_5 \vee x_8) \oplus (x_5 \vee x_9) \oplus (x_5 \vee x_{10}) \oplus (x_6 \vee x_7) \oplus (x_6 \vee x_8) \oplus (x_6 \vee x_9) \oplus (x_6 \vee x_{10}) \oplus (x_7 \vee x_8) \oplus (x_7 \vee x_9) \oplus (x_7 \vee x_{10}) \oplus (x_8 \vee x_9) \oplus (x_8 \vee x_{10}) \oplus (x_9 \vee x_{10}) \oplus (x_1 \vee x_2 \vee x_3) \oplus (x_1 \vee x_2 \vee x_4) \oplus (x_1 \vee x_2 \vee x_5) \oplus (x_1 \vee x_2 \vee x_6) \oplus (x_1 \vee x_2 \vee x_7) \oplus (x_1 \vee x_2 \vee x_8) \oplus (x_1 \vee x_2 \vee x_9) \oplus (x_1 \vee x_2 \vee x_{10}) \oplus (x_1 \vee x_3 \vee x_4) \oplus (x_1 \vee x_3 \vee x_5) \oplus (x_1 \vee x_3 \vee x_6) \oplus (x_1 \vee x_3 \vee x_7) \oplus (x_1 \vee x_3 \vee x_8) \oplus (x_1 \vee x_3 \vee x_9) \oplus (x_1 \vee x_3 \vee x_{10}) \oplus (x_1 \vee x_4 \vee x_5) \oplus (x_1 \vee x_4 \vee x_6) \oplus (x_1 \vee x_4 \vee x_7) \oplus (x_1 \vee x_4 \vee x_8) \oplus (x_1 \vee x_4 \vee x_9) \oplus (x_1 \vee x_4 \vee x_{10}) \oplus (x_1 \vee x_5 \vee x_6) \oplus (x_1 \vee x_5 \vee x_7) \oplus (x_1 \vee x_5 \vee x_8) \oplus (x_1 \vee x_5 \vee x_9) \oplus (x_1 \vee x_5 \vee x_{10}) \oplus (x_1 \vee x_6 \vee x_7) \oplus (x_1 \vee x_6 \vee x_8) \oplus (x_1 \vee x_6 \vee x_9) \oplus (x_1 \vee x_6 \vee x_{10}) \oplus (x_1 \vee x_7 \vee x_8) \oplus (x_1 \vee x_7 \vee x_9) \oplus (x_1 \vee x_7 \vee x_{10}) \oplus (x_1 \vee x_8 \vee x_9) \oplus (x_1 \vee x_8 \vee x_{10}) \oplus (x_1 \vee x_9 \vee x_{10}) \oplus (x_2 \vee x_3 \vee x_4) \oplus (x_2 \vee x_3 \vee x_5) \oplus (x_2 \vee x_3 \vee x_6) \oplus (x_2 \vee x_3 \vee x_7) \oplus (x_2 \vee x_3 \vee x_8) \oplus (x_2 \vee x_3 \vee x_9) \oplus (x_2 \vee x_3 \vee x_{10}) \oplus (x_2 \vee x_4 \vee x_5) \oplus (x_2 \vee x_4 \vee x_6) \oplus (x_2 \vee x_4 \vee x_7) \oplus (x_2 \vee x_4 \vee x_8) \oplus (x_2 \vee x_4 \vee x_9) \oplus (x_2 \vee x_4 \vee x_{10}) \oplus (x_2 \vee x_5 \vee x_6) \oplus (x_2 \vee x_5 \vee x_7) \oplus (x_2 \vee x_5 \vee x_8) \oplus (x_2 \vee x_5 \vee x_9) \oplus (x_2 \vee x_5 \vee x_{10}) \oplus (x_2 \vee x_6 \vee x_7) \oplus (x_2 \vee x_6 \vee x_8) \oplus (x_2 \vee x_6 \vee x_9) \oplus (x_2 \vee x_6 \vee x_{10}) \oplus (x_2 \vee x_7 \vee x_8) \oplus (x_2 \vee x_7 \vee x_9) \oplus (x_2 \vee x_7 \vee x_{10}) \oplus (x_2 \vee x_8 \vee x_9) \oplus (x_2 \vee x_8 \vee x_{10}) \oplus (x_2 \vee x_9 \vee x_{10}) \oplus (x_3 \vee x_4 \vee x_5) \oplus (x_3 \vee x_4 \vee x_6) \oplus (x_3 \vee x_4 \vee x_7) \oplus (x_3 \vee x_4 \vee x_8) \oplus (x_3 \vee x_4 \vee x_9) \oplus (x_3 \vee x_4 \vee x_{10}) \oplus (x_3 \vee x_5 \vee x_6) \oplus (x_3 \vee x_5 \vee x_7) \oplus (x_3 \vee x_5 \vee x_8) \oplus (x_3 \vee x_5 \vee x_9) \oplus (x_3 \vee x_5 \vee x_{10}) \oplus (x_3 \vee x_6 \vee x_7) \oplus (x_3 \vee x_6 \vee x_8) \oplus (x_3 \vee x_6 \vee x_9) \oplus (x_3 \vee x_6 \vee x_{10}) \oplus (x_3 \vee x_7 \vee x_8) \oplus (x_3 \vee x_7 \vee x_9) \oplus (x_3 \vee x_7 \vee x_{10}) \oplus (x_3 \vee x_8 \vee x_9) \oplus (x_3 \vee x_8 \vee x_{10}) \oplus (x_3 \vee x_9 \vee x_{10}) \oplus (x_4 \vee x_5 \vee x_6) \oplus (x_4 \vee x_5 \vee x_7) \oplus (x_4 \vee x_5 \vee x_8) \oplus (x_4 \vee x_5 \vee x_9) \oplus (x_4 \vee x_5 \vee x_{10}) \oplus (x_4 \vee x_6 \vee x_7) \oplus (x_4 \vee x_6 \vee x_8) \oplus (x_4 \vee x_6 \vee x_9) \oplus (x_4 \vee x_6 \vee x_{10}) \oplus (x_4 \vee x_7 \vee x_8) \oplus (x_4 \vee x_7 \vee x_9) \oplus (x_4 \vee x_7 \vee x_{10}) \oplus (x_4 \vee x_8 \vee x_9) \oplus (x_4 \vee x_8 \vee x_{10}) \oplus (x_4 \vee x_9 \vee x_{10}) \oplus (x_5 \vee x_6 \vee x_7) \oplus (x_5 \vee x_6 \vee x_8) \oplus (x_5 \vee x_6 \vee x_9) \oplus (x_5 \vee x_6 \vee x_{10}) \oplus (x_5 \vee x_7 \vee x_8) \oplus (x_5 \vee x_7 \vee x_9) \oplus (x_5 \vee x_7 \vee x_{10}) \oplus (x_5 \vee x_8 \vee x_9) \oplus (x_5 \vee x_8 \vee x_{10}) \oplus (x_5 \vee x_9 \vee x_{10}) \oplus (x_6 \vee x_7 \vee x_8) \oplus (x_6 \vee x_7 \vee x_9) \oplus (x_6 \vee x_7 \vee x_{10}) \oplus (x_6 \vee x_8 \vee x_9) \oplus (x_6 \vee x_8 \vee x_{10}) \oplus (x_6 \vee x_9 \vee x_{10}) \oplus (x_7 \vee x_8 \vee x_9) \oplus (x_7 \vee x_8 \vee x_{10}) \oplus (x_7 \vee x_9 \vee x_{10}) \oplus (x_8 \vee x_9 \vee x_{10})$$

In this formula $\vee$ stands for logical or, and $\oplus$ stands for exclusive or (xor). Remember that in C++ and Java these two binary operators are denoted as "`||`" and "`^`".

Given the values of $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$, calculate the value of $f(x_1, x_2, \ldots, x_{10})$.

## Input

The input file contains 10 numbers $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$, and $x_{10}$. Each of them is either 0 or 1.

## Output

Output a single value — $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$.

## Example

| explicit.in | explicit.out |
|---|---|
| 1 0 0 1 0 0 1 0 0 1 | 0 |

# Problem F. Flat

| | |
|---|---|
| Input file: | `flat.in` |
| Output file: | `flat.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

You are one of the developers of software for a real estate agency. One of the functions you are to implement is calculating different kinds of statistics for flats the agency is selling. Each flat consists of different types of rooms: bedroom, bathroom, kitchen, balcony and others.

The cost of the flat is equal to the product of reduced total area and the cost of one square metre. Reduced total area is the total area of all rooms except for balconies plus one half of balconies total area.

You will be given some information about the area of each room in the flat and the cost of one square metre. You are to calculate the following values for the flat:

- the total area of all rooms;
- the total area of all bedrooms;
- the cost of the flat.

## Input

The first line of the input file contains two integer numbers $n$ ($1 \le n \le 10$) and $c$ ($1 \le c \le 100\,000$) — number of rooms in the flat and the cost of one square metre, respectively.

Each of the following $n$ lines contains an integer number $a_i$ ($1 \le a_i \le 100$) and a word $t_i$ — the area of $i$-th room and its type, respectively. Word $t_i$ is one of the following: "`bedroom`", "`bathroom`", "`kitchen`", "`balcony`", "`other`".

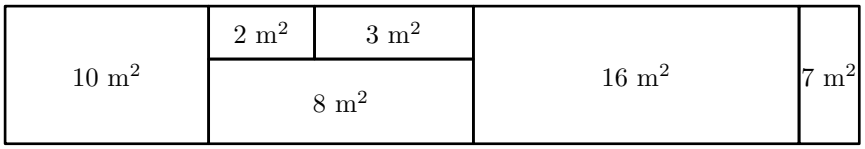
## Output

The first line of the output file should contain one integer number — the total area of all rooms of the flat. The second line of the output file should contain one integer number — the total area of bedrooms of the flat. The third line of the output file should contain one real number — the cost of the flat with precision not worse than $10^{-6}$.

## Examples

| flat.in | flat.out |
|---|---|
| 6 75000<br>8 other<br>3 bathroom<br>2 bathroom<br>10 kitchen<br>16 bedroom<br>7 balcony | 46<br>16<br>3187500 |
| 2 75123<br>10 kitchen<br>15 balcony | 25<br>0<br>1314652.5 |

The figure shows the flat from the first example.



# Problem G. Garage

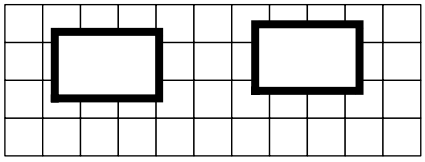
| | |
|---|---|
| Input file: | `garage.in` |
| Output file: | `garage.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Wow! What a lucky day! Your company has just won a social contract for building a garage complex. Almost all formalities are done: contract payment is already transferred to your account.

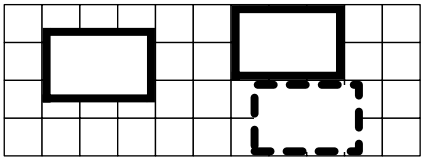
So now it is the right time to read the contract. Okay, there is a sandlot in the form of $W \times H$ rectangle and you have to place some garages there. Garages are $w \times h$ rectangles and their edges must be parallel to the corresponding edges of the sandlot (you may not rotate garages, even by 90°). The coordinates of garages may be non-integer.

You know that the economy must be economical, so you decided to place as *few* garages as possible. Unfortunately, there is an opposite requirement in the contract: placing maximum possible number of garages.

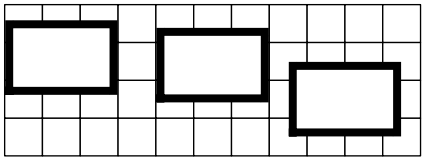
Now let's see how these requirements are checked... The plan is accepted if it is impossible to add a new garage without moving the other garages (the new garage must also have edges parallel to corresponding sandlot edges).



Accepted optimal plan



Rejected plan



Accepted, but non-optimal plan

Time is money, find the minimal number of garages that must be ordered, so that you can place them on the sandlot and there is no place for an extra garage.

## Input

The only line contains four integers: $W$, $H$, $w$, $h$ — dimensions of sandlot and garage in meters. You may assume that $1 \le w \le W \le 30\,000$ and $1 \le h \le H \le 30\,000$.

## Output

Output the optimal number of garages.

## Examples

| garage.in | garage.out |
|---|---|
| 11 4 3 2 | 2 |
| 10 8 3 4 | 2 |
| 15 7 4 2 | 4 |

The plan on the first picture is accepted and optimal for the first example. Note that a rotated $(2 \times 3)$ garage could be placed on the sandlot, but it is prohibited by the contract.

# Problem H. Homo or Hetero?

| | |
|---|---|
| Input file: | `homo.in` |
| Output file: | `homo.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Consider a list of numbers with two operations:

- `insert number` — adds the specified number to the end of the list.
- `delete number` — removes the first occurrence of the specified number from the list. If the list does not contain the number specified, no changes are performed.

For example: the result of the insertion of a number 4 to the list $[1, 2, 1]$ is the list $[1, 2, 1, 4]$. If we delete the number 1 from this list, we get the list $[2, 1, 4]$, but if we delete the number 3 from the list $[1, 2, 1, 4]$, the list stays unchanged.

The list is *homogeneous* if it contains at least two equal numbers and the list is *heterogeneous* if it contains at least two different numbers. For example: the list $[2, 2]$ is homogeneous, the list $[2, 1, 4]$ is heterogeneous, the list $[1, 2, 1, 4]$ is both, and the empty list is neither homogeneous nor heterogeneous.

Write a program that handles a number of the operations `insert` and `delete` on the empty list and determines list's homogeneity and heterogeneity after each operation.

## Input

The first line of the input file contains an integer number $n$ — the number of operations to handle ($1 \le n \le 100\,000$).

Following $n$ lines contain one operation description each. The operation description consists of a word "`insert`" or "`delete`", followed by an integer number $k$ — the operation argument ($-10^9 \le k \le 10^9$).

## Output

For each operation output a line, containing a single word, describing the state of the list after the operation:

- "`both`" — if the list is both homogeneous and heterogeneous.
- "`homo`" — if the list is homogeneous, but not heterogeneous.
- "`hetero`" — if the list is heterogeneous, but not homogeneous.
- "`neither`" — if the list is neither homogeneous nor heterogeneous.

## Example

| homo.in | homo.out |
|---|---|
| 11<br>insert 1<br>insert 2<br>insert 1<br>insert 4<br>delete 1<br>delete 3<br>delete 2<br>delete 1<br>insert 4<br>delete 4<br>delete 4 | neither<br>hetero<br>both<br>both<br>hetero<br>hetero<br>hetero<br>neither<br>homo<br>neither<br>neither |

# Problem I. Java vs C++

| | |
|---|---|
| Input file: | `java_c.in` |
| Output file: | `java_c.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Apologists of Java and C++ can argue for hours proving each other that their programming language is the best one. Java people will tell that their programs are clearer and less prone to errors, while C++ people will laugh at their inability to instantiate an array of generics or tell them that their programs are slow and have long source code.

Another issue that Java and C++ people could never agree on is identifier naming. In Java a multiword identifier is constructed in the following manner: the first word is written starting from the small letter, and the following ones are written starting from the capital letter, no separators are used. All other letters are small. Examples of a Java identifier are `javaIdentifier`, `longAndMnemonicIdentifier`, `name`, `nEERC`.

Unlike them, C++ people use only small letters in their identifiers. To separate words they use underscore character '_'. Examples of C++ identifiers are `c_identifier`, `long_and_mnemonic_identifier`, `name` (you see that when there is just one word Java and C++ people agree), `n_e_e_r_c`.

You are writing a translator that is intended to translate C++ programs to Java and vice versa. Of course, identifiers in the translated program must be formatted due to its language rules — otherwise people will never like your translator.

The first thing you would like to write is an identifier translation routine. Given an identifier, it would detect whether it is Java identifier or C++ identifier and translate it to another dialect. If it is neither, then your routine should report an error. Translation must preserve the order of words and must only change the case of letters and/or add/remove underscores.

## Input

The input file consists of one line that contains an identifier. It consists of letters of the English alphabet and underscores. Its length does not exceed 100.

## Output

If the input identifier is Java identifier, output its C++ version. If it is C++ identifier, output its Java version. If it is none, output "`Error!`" instead.

## Example

| java_c.in | java_c.out |
|---|---|
| `long_and_mnemonic_identifier` | `longAndMnemonicIdentifier` |
| `anotherExample` | `another_example` |
| `i` | `i` |
| `bad_Style` | `Error!` |

# Problem J. John's Inversions

| | |
|---|---|
| Input file: | `john.in` |
| Output file: | `john.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

John has recently come across the definition of an inversion.

An *inversion* in a sequence of numbers $s_k$ is any pair $s_i$, $s_j$ such that $i < j$ and $s_i > s_j$.

John believes that inversions are a perfect tool for estimation of how well a sequence of numbers is sorted. The smaller the number of inversions in the sequence, the better it is sorted. For example, the sequences sorted in the ascending order have zero inversions in them.

Peter gave John a stack of $n$ cards. Each card has two numbers written on it — one in red and one in blue color. John is anxious to apply his knowledge of inversions to that stack.

He places the cards in front of him in arbitrary order and calculates the total number of *nice inversions* in front of him. John considers an inversion to be nice if it consists of the numbers of the same color. In our case nice inversion can be formed by either two blue or two red numbers. If the number of nice inversions is too big by John's standards, he rearranges the cards and repeats the process.

Your task is to help John find out the minimal possible number of nice inversions he can get while following his algorithm.

## Input

The first line of the input file contains one integer number $n$ — the number of cards in the deck ($1 \le n \le 100\,000$). The following $n$ lines describe one card each. The $i$-th line contains two integer numbers $r_i$ and $b_i$ ($1 \le r_i, b_i \le 10^9$) — the numbers written on $i$-th card in red and blue colors respectively.

## Output

Output should contain exactly one integer number — the minimal possible number of nice inversions.

## Examples

| john.in | john.out |
|---|---|
| 3<br>10 3<br>20 2<br>30 1 | 3 |
| 4<br>2 2<br>5 25<br>2 1<br>10 9 | 1 |