Exercise: Multidimensional Lists

Problems for exercise and homework for the [Python Advanced Course @SoftUni.](https://softuni.bg/trainings/4106/python-advanced-may-2023)

Submit your solutions in the SoftUni judge system at <https://judge.softuni.org/Contests/3194>.

# Flatten Lists

Write a program to flatten **several lists** of numbers received in the following format:

* + String with numbers or empty strings separated by **"|"**
  + Values are separated by spaces (**" "**, one or several)
  + Order the output list from the **last** to the **first matrix sub-lists** and their values from **left** to **right** as shown below

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 |4 5 6 | 7 88 | 7 88 4 5 6 1 2 3 |
| 7 | 4 5|1 0| 2 5 |3 | 3 2 5 1 0 4 5 7 |
| 1| 4 5 6 7 | 8 9 | 8 9 4 5 6 7 1 |

# Matrix Modification

Write a program that **reads a matrix** from the console and **changes its values**. On the first line, you will get the matrix's **rows - N**. You will get elements for each **column** on the following **N** lines, separated with a **single space**. You will be receiving commands in the following format:

* **"Add {row} {col} {value}"** – **Increase** the number at the given **coordinates** with the **value.**
* **"Subtract {row} {col} {value}"** – **Decrease** the number at the given **coordinates** by the **value**.

If **the coordinate is invalid**, you should **print "Invalid coordinates"**. A coordinate **is valid** if both of the given indexes are in range **[0; len() – 1]**.

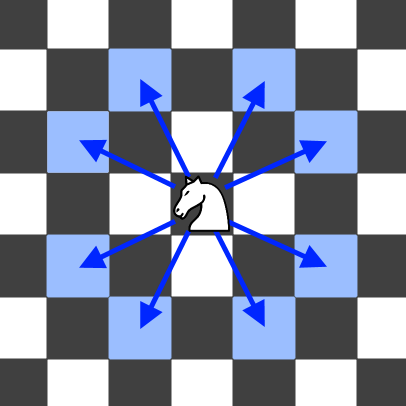
When you receive **"END"**, you should **print the matrix** and **stop the program**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  1 2 3  4 5 6  7 8 9  Add 0 0 5  Subtract 1 1 2 END | 6 2 3  4 3 6  7 8 9 |
| 4  1 2 3 4  5 6 7 8 | Invalid coordinates Invalid coordinates  -41 2 3 4 |

|  |  |
| --- | --- |
| 8 7 6 5  4 3 2 1  Add 4 4 100  Add 3 3 100  Subtract -1 -1 42  Subtract 0 0 42 END | 5 6 7 8  8 7 6 5  4 3 2 101 |

# Knight Game



Chess is the oldest game, but it is still popular these days. You will use only one chess piece for this task - the **Knight**.

A chess knight has **8 possible moves** it can make, as illustrated. It can move to the **nearest** square but **not on the same** [**row**](https://en.wikipedia.org/wiki/Glossary_of_chess#rank), [**column**](https://en.wikipedia.org/wiki/Glossary_of_chess#file), or [**diagonal**](https://en.wikipedia.org/wiki/Glossary_of_chess#diagonal). (e.g., it can move two squares horizontally, then one square vertically, or it can move one square horizontally then two squares vertically - i.e., in an "**L**" pattern.)

The knight game is played on a board with dimensions **N x N**.

You will receive a board with **"K"** for knights and "**0"** for empty cells. Your task is to **remove knights** until **no knights that can attack one another** with one move **are left**.

Always **remove** the knight who **can attack the greatest number of knights**. If there are **two or more knights** with the same number of attacks, remove the **top-left one**.

## Input

* On the first line, you will receive integer **N** - the size of the board
* On the following **N** lines, you will receive strings with **"K"** and "**0"**

## Output

* Print a **single integer** with the **number of knights that need to be removed**.

## Constraints

* The size of the board will be **0 < N < 30**
* Time limit: **0.3 sec**. Memory limit: **16 MB**

**Examples**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  |  |  |  |
| 5 | 1 | 2 | 0 | 8 | 12 |
| 0K0K0 |  | KK |  | 0K0KKK00 |  |
| K000K |  | KK |  | 0K00KKKK |  |
| 00K00 |  |  |  | 00K0000K |  |
| K000K |  |  |  | KKKKKK0K |  |
| 0K0K0 |  |  |  | K0K0000K |  |
|  |  |  |  | KK00000K |  |
|  |  |  |  | 00K0K000 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | 000K00KK |  |

# Easter Bunny

*Your task is to collect as many eggs as possible.*

On the first line, you will be given a **number** representing the **size of the field**. In the following few lines, you will be given a **field** with:

* **One bunny** - randomly placed in it and marked with the symbol **"B"**
* **Number** of eggs placed at different positions of the field and **traps** marked with **"X"**

Your job is to determine the direction in which the bunny should go to collect the **maximum** number of eggs. The directions that should be considered as possible are **up, down, left,** and **right.** If you reach a **trap** while checking some of the directions, you should **not** consider the fields after the trap in this direction. The bunny can move within the field and cannot go outside its boundaries. Do **not** consider **negative indices** as valid ones. For more clarifications, see the examples below.

Note: In some directions, the collected eggs can happen to be **zero** or a **negative** number.

## Input

* **A number** representing the size of the field
* **The matrix** representing the field (each position **separated by a single space**)

## Output

* **The direction** which should be considered as **best (lowercase)**
* The field **positions** from which we are **collecting eggs as lists**
* The **total** number of eggs collected

## Constraints

* There will **NOT** be two or more paths consisting of the same total amount of eggs.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 5 | right | The number of eggs if the bunny goes up is equal to 7. If it goes |
| 1 3 7 9 11 | [3, 1] | down = 9, there are no eggs on the left and 87 on the right. That's |
| X 5 4 X 63 | [3, 2] | why the bunny should follow this direction (right) and collect the |
| 7 3 21 95 1  B 1 73 4 9 | [3, 3]  [3, 4] | eggs provided there. |
| 9 2 33 2 0 | 87 |  |
| 8 | down |  |
| 4 18 9 7 24 41 52 11 | [6, 2] |
| 54 21 19 X 6 34 75 57 | [7, 2] |
| 76 67 7 44 76 27 56 37 | 113 |
| 92 35 25 37 52 34 56 72 |  |
| 35 X 1 45 4 X 37 63 |  |
| 105 X B 2 12 43 5 19 |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 48 | 19 | 35 | 20 | 32 | 27 | 42 4 |  |  |
| 73 | 88 | 78 | 32 | 37 | 52 | X 22 |

# Alice in Wonderland

*Alice is going to the mad tea party, to see her friends. On the way to the party, she needs to collect bags of tea.*

You will be given an integer **n** for the **size** of the Wonderland territory with a **square** shape. On the following **n** lines, you will receive the **rows** of the territory:

* Alice will be placed in a **random position**, marked with the letter "**A**".
* On the territory, there will be bags of tea, represented as numbers. If Alice **steps on a number position**, she collects the tea bags and **increases the quantity with the corresponding number**.
* There will **always** be **one rabbit hole** on the territory **marked** with the **letter** "**R**".
* **All of the empty positions** will be marked with **"."**.

After the field state, you will be given **commands** for **Alice's movements**. Move commands can be: "**up**", "**down**", "**left**" or "**right**".

When Alice collects **at least 10 bags of tea, she is ready to go to the tea party,** and she does **not need** to continue collecting. Otherwise, **if she steps into the rabbit hole** or **goes out of the territory**, she **can't return,** and the program **ends**.

In the end, the path she walked had to be marked with **'\*'**. For more clarifications, see the examples below.

## Input

* + On the first line, you will be given the integer **n** – the size of the **square** matrix
  + On the following n lines **- matrix** representing the field (each position **separated by a single space**)
  + On each of the following lines, you will be given a move command

## Output

* + On the first line:
    - If Alice steps into the rabbit hole or goes out of the territory, print:

### "Alice didn't make it to the tea party."

* + - If she collected at least 10 bags of tea, print:

### "She did it! She went to the party."

* + On the following lines, print the matrix.

## Constraints

* + Alice will **always** either **go outside Wonderland** or **collect 10 bags of tea**
  + All the commands will be valid
  + All of the given **numbers** will be valid **integers** in the range [0, 10]

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |

|  |  |
| --- | --- |
| 5  . A . . 1  R . 2 . . 4 7 . 1 .  . . . 2 .  . 3 . . .  down right left down up  left | Alice didn't make it to the tea party.  . \* . . 1  \* \* \* . .  4 \* . 1 .  . . . 2 .  . 3 . . . |
| 7  . A . 1 1 . .  9 . . . 6 . 5  . 6 . R . . .  . 3 . . 1 . .  . . . 2 . . 2  . 3 . . 1 . .  . 8 3 . . . 2  left down down  right | She did it! She went to the party.  \* \* . 1 1 . .  \* . . . 6 . 5  \* \* . R . . .  . 3 . . 1 . .  . . . 2 . . 2  . 3 . . 1 . .  . 8 3 . . . 2 |

# Range Day

*You are participating in a Firearm course. It is a training day at the shooting range.*

You will be given a **matrix with 5 rows and 5 columns**. It is a **shotgun range** represented as some **symbols** separated by a **single space**:

* **Your position is** marked with the symbol "**A**"
* **Targets** marked with the symbol "**x**"
* **All of the empty positions** will be marked with "**.**"

After the field state, you will be given an integer representing the **number of commands** you will receive. The possible commands are:

* **"move {right/left/up/down} {steps}"** – you should **move** in the given **direction** with the given **steps**. **You can only move** if the field you want to **step on is marked with "."**.
* **"shoot {right/left/up/down}"** – you should **shoot** in the given direction (from your **current position without moving**). Beware that there might be targets that **stand in the way** of other targets, and you **cannot reach** them - you **can shoot** only **the nearest** target. **When you have shot a target, the field becomes an empty position (".").**

**Validate** the positions since they can be **outside** the field. Keep track of all the **shot targets**:

* If at any point there are **no targets left**, **end** the program and print: **"Training completed! All**

### {count\_targets} targets hit.".

* If, after you perform all **the commands,** there are some **targets left** print: **"Training not completed!**

### {count\_left\_targets} targets left.".

Finally, print the **index positions** of the **targets that you hit,** as shown in the examples.

**Input**

* **5 lines** representing the field (symbols, **separated by a single space**)
* **N** - count of **commands**
* On the following **N lines** - the commands in the format described above

## Output

* On the **first line,** print one of the following:
  + If all the **targets** were **shot**

### "Training completed! All {count\_targets} targets hit."

* + Otherwise:

### "Training not completed! {count\_left\_targets} targets left."

* Finally, print the **index positions "[{row}, {column}]"** of the **targets that you hit**, as shown in the examples.

## Constraints

* All the **commands** will be **valid**
* There will **always be at least one target**

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| . . . . .  x . . . .  . A . . .  . . . x .  . x . . x 3  shoot down move right 4  move left 1 | Training not completed! 3 targets left. [4, 1] |
| . . . . .  . . . . .  . A x . .  . . . . . | Training completed! All 2 targets hit. [4, 1]  [2, 2] |

|  |  |
| --- | --- |
| . x . . . 2  shoot down shoot right |  |
| . . . . . | Training not completed! 1 targets left. |
| . . . . . | [4, 1] |
| . . x . . |  |
| . . . . . |  |
| . x . . A |  |
| 3 |  |
| shoot down |  |
| move right 2 |  |
| shoot left |  |

# Present Delivery

*The presents are ready, and Santa has to deliver them to the kids.*

You will receive an integer **m** for the **number** of **presents** Santa has and an integer **n** for the **size** of the **neighborhood**

with a **square** shape. On the following lines, you will receive the **matrix**, which represents the neighborhood.

Santa will be in a **random cell**, marked with the letter **"S"**. Each cell stands for a house where children may live. If the cell has an **"X"** on it, that means there lives a **naughty** kid. Otherwise, if a **nice** kid lives there, the cell is marked with **"V"**. There can also be cells marked with **"C"** for cookies. **All of the empty positions** will be marked with **"-"**.

Santa can move "**up**", **"down"**, **"left"**, **"right"** with **one position** each time. These will be the **commands** that you receive. If he moves to a house with a **nice** kid, the kid **receives a present**, but if Santa reaches a house with a **naughty** kid, he **doesn't** drop a present. If the command sends Santa to a cell marked with **"C"**, Santa eats cookies and becomes happy and extra generous to **all the kids around him**\* (meaning all of them will receive presents - it doesn't matter if naughty or nice). If Santa has been to a house, the cell becomes **"-"**.

**Note**: \*around him means on his left, right, upwards, and downwards by one cell. In this case, **Santa** doesn't move to these cells, or if he does, he **returns** to the **cell** where the **cookie** was.

If Santa runs out of presents **or** receives the command "**Christmas morning**", you should end the program. Keep in mind that you should check whether all the nice kids received presents.

## Input

* On the first line, you are given the integer **m** - the count of presents
* On the second - integer **n** - the size of the neighborhood
* The **following n lines** hold the values for every **row**
* On each of the following lines you will get a command

## Output

* On the first line:
  + If Santa runs out of presents, **but** there are still some **nice** kids left print: "**Santa ran out of presents!**"
* Next, print the matrix.
* In the end, print one of these messages:
  + If he manages to give **all** the nice kids presents, print:

### "Good job, Santa! {count\_nice\_kids} happy nice kid/s."

* + Otherwise, print:

**"No presents for {count nice kids} nice kid/s."**

## Constraints

* The size of the **square** matrix will be between **[2…10].**
* Santa's position will be marked with an '**S**'.
* There will **always** be **at least 1 nice** kid.
* There **won't be a case** where the cookie is on the border of the matrix.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  4   * X V - * S - V   - - - -  X - - -  up right down right  Christmas morning | - - - -  - - - S  - - - -  X - - -  Good job, Santa! 2 happy nice kid/s. | Santa has 5 presents. The size of the matrix is 4. After we receive the matrix, we start reading commands. The first one is "up". The "X" means there is a naughty kid, so Santa moves on without dropping any presents. Next, he reaches a nice kid and drops a present. The "down" command moves Santa to an empty cell. The last command before the "Christmas morning" message is "right". Again we have a nice kid. The count of nice kids reached 2, and we don't have any nice kids without presents left. So we print the appropriate message. |
| 3  4  - - - - V - X -  - V C V  - - - S  left up | Santa ran out of presents!  - - - -  V - - -  - - S -  - - - -  No presents for 1 nice kid/s. | The commands send Santa to a cell with a cookie, so all of the kids around him receive presents. He runs out of presents because we have 3 kids there and only 3 presents. The program ends, and we have 1 nice kid that hasn't received a present. |