**Problem 1 – Enigma**

You are given **N** lines of encrypted messages. The messages will contain ASCII characters. In each message, only the **Latin letters and special characters will be encrypted**. The numbers and whitespace will not be encrypted. Your task is to write a program to decrypt the messages. The formula for the decrypting each letter is **X = k + m**, where **X** is the ASCII code of the decrypted letter, **k** is the ASCII code of the encrypted character and **m** is the **integer** **half of the length** of the input line, **without the numbers and whitespace**. (Hint: length()/2)

**Input**

The input comes from the console. The first line holds the **count** **N**. After that there are **N lines** with the encrypted messages.

The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

Print at the console the decrypted messages, each on **separate line.**

Each message should hold the numbers, whitespace and decrypted letters. See the examples below.

**Constraints**

* The **count** **N** will be an integer number in the range [1…50].
* The input **lines length** will be an integer number in the range [1...35].
* The input **lines** may hold **any ASCII character.**
* Time limit: 0.2 sec. Memory limit: 16 MB.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  Ie\jkd\_ ^Wi =h[Wj I[Yh[j | SoftUni has Great Secret  *The length without whitespaces is 21. Integer division 21/2 = 10. ASCII(I) = 73. 73 + 10 = 83. 83 = ASCII(S). . ASCII(e) = 101. 101 + 10 = 111. 111 = ASCII(o).* |
| **Input** | **Output** |
| 3  Tbi`ljb  rm rfc  grkdib | Welcome  to the  jungle |
| **Input** | **Output** |
| 1  P^ aZo^ 350 fbllbe^l | We have 350 missiles |

# Problem 2 – Magic Card

Sashko loves to play card games. He even invented his own game. The game uses a **standard deck of 52 cards**. The card faces are: **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **J**, **Q**, **K** and **A**. The cards suits are denoted by the letters **S** (spades), **H** (hearts), **D** (diamonds) and **C** (clubs). The player is given a hand of cards, a string (“**odd**” or “**even**”), and a **magic card**. You need to **count the sum of all cards at odd or even positions (positions start from 0)**. Card **values** are the following: 2 -> 20, 3 -> 30, 4 -> 40, 5 -> 50, 6 -> 60, 7 -> 70, 8 -> 80, 9 -> 90, 10 -> 100, J -> 120, Q -> 130, K -> 140, A -> 150. When a card’s suit is the same as the suit of the magic card its value is **doubled**. When a card’s face is the same as the face of the magic card its value is **tripled**. The input hand **will not contain** the magic card.

For example, if Sashko gets the hand "**2C 2D 2H AS 10H 10C 2S 3S 5D KD**", the string “**odd**” and a magic card “**AD**”. The value of the hand is 20 \* 2 + 150 \* 3 + 100 + 30 + 140 \* 2 = 900.

Write a program that takes a hand of cards and counts the sum.

### Input

The input comes from the console**.** The first line **is holding the hand of cards**. Cards are separated by a space.

The second line **is holding a string – “odd” or “even”**. The third line **is holding the magic card**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console a single number: the **value of the hand**.

### Constraints

* The **count** of the cards will be in the range [1…99].
* **Card faces** will beone of the following values: [**2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **J**, **Q**, **K**, **A**].
* **Card suits** will beone of the following values: [**S**, **H**, **D**, **C**].
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2C 2D 2H AS 10H 10C 2S 3S 5D KD  odd  AD | 900 |
| AS KH 10C  even  KD | 250 |
| AS 10C KS KH KD 9H JH QS 3H QD QH 8S 10D 10S 7C JD  even  3D | 1180 |

# Problem 3 – Labyrinth Dash

Enough hard problems. Let’s play a game! You will be given the layout of a labyrinth (a two-dimensional array) and a series of moves. Your task is to navigate the labyrinth and **print the outcome of each move**.

On the first line of input you will be given the **number N, representing the count of rows** of the labyrinth. On each of the next N lines you will receive a **string, containing the layout of the given row**. On the last line of input you will receive **a string, containing the moves** you need to make. Each move will be one of the following symbols: **“v” (move down), “^” (move up), “<” (move left) or “>” (move right).** The **player starts with 3 lives and begins the journey at position (0, 0).** When you make a move, there can be several different outcomes:

1. **Hit a wall** – a wall is represented by the symbols **“\_” (underscore) and “|” (pipe).** Hitting a wall means the player stays in place; in this case you should print on the console **“Bumped a wall.”**
2. **Land on an obstacle** – obstacles are the following symbols: **“@”, “#”, “\*”**. If you move to a position containing one of these symbols the player loses a life point and you should print **“Ouch! That hurt! Lives left: X”** on the console. If the player is left with 0 lives, the game ends and you should print also **“No lives left! Game Over!”**
3. **Get a new life** – when you land on the symbol **“$”** the player receives an additional life point.  
   Print **“Awesome! Lives left: X”** on the console. Additional lives (‘$’) are removed once the player passes through the cell (i.e. they are replaced with dots).
4. **Drop out of the labyrinth** – if you land on an empty cell (one containing a space), or outside the boundaries of the array, the game ends and you should print **“Fell off a cliff! Game Over!”**
5. **Land on the symbol “.” (dot)** – move the player to the new position, nothing else happens; print on the console **“Made a move!”**

When the game ends (either the player has lost or all moves were made), print **“Total moves made: X”**.

### Input

* The input data should be read from the console.
* On the first line of input you will receive the number N – number of rows of the labyrinth.
* On the next N lines you will receive the layout of the labyrinth.
* On the last line you will receive the moves you need to make as a string.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console.
* For each outcome print the required output as described above.

### Constraints

* The number N will be an integer in the range [1 … 15].
* The labyrinth will contain only the symbols – “\_”, “|”, “@”, “#”, “\*”, “$”, “ “ (single whitespace), “.”.
* The string containing the moves will contain only the symbols – “v”, “^”, “<”, and “>”.
* Allowed working time for your program: 0.5 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  .|  ..|  \*.$ .  ###...  \_\_\_\_\_\_  >v>>vv>>>^^^<< | Bumped a wall.  Made a move!  Made a move!  Bumped a wall.  Made a move!  Ouch! That hurt! Lives left: 2  Ouch! That hurt! Lives left: 1  Made a move!  Made a move!  Fell off a cliff! Game Over!  Total moves made: 8 | Player starts at (0, 0). First move is “>” (right), which takes the player into a wall. Next, he moves down and right. The next move is right again and he hits another wall. He then moves down twice, on the second move he lands on an obstacle (“#”) and loses a life point. He then moves right and loses another life. Two moves to the right are followed by a move upwards which takes him out of the labyrinth (empty cell), so the game is over. The total number of moves where the player actually changed position is 8. |

# Problem 4 – Weightlifting

Soon there will be a world championship on Weightlifting and all the famous players are preparing for it. As a coach of an international team, your job is to ensure each player has sufficient training, but is not exhausted more than he should. Every day you prepare a list of exercises and at the end of the day you want to know how much the players have trained. The list will be a sequence of **N** lines holding the exercises of the players in your team for that day.

Write a program that prints **all players** **in alphabetical order**. For each player, the program should print the **type of exercise** and the total weight the player has lifted doing that exercise. The **exercises** should be **ordered alphabetically** for each player.

### Input

The input comes from the console. The first line holds the length of the input list **N**.

On the next N lines, there will be information about the exercises for that day in the following format: “**<player> <exercise> <weight> kg”**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console **one line for each player** in the following format: **<player> : <exercise> - <weight> kg, <exercise> - <weight> kg,…**

**Constraints**

* The **count N** will be an integer number in the range [1…50].
* The **weight** will be a positive integer number.
* The input **lines** may hold **any ASCII character.**
* Time limit: 0.2 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  Jack Squat 200 kg  John Deadlift 220 kg  John Squat 150 kg  Jack Deadlift 120 kg  John Benchpress 200 kg  John Squat 140 kg | Jack : Deadlift - 120 kg, Squat - 200 kg  John : Benchpress 200 kg, Deadlift - 220 kg, Squat - 290 kg |
| **Input** | **Output** |
| 5  Mark Squat 500 kg  John Deadlift 220 kg  Mark Squat 50 kg  Mark Deadlift 120 kg  John Squat 200 kg | John : Deadlift - 220 kg, Squat - 200 kg  Mark : Deadlift - 120 kg, Squat - 550 kg |