Bombs

Ezio is still learning how to make bombs. With their help, he will save civilization. We should help Ezio to make his perfect bombs.

You will be given **two sequences of integers, representing bomb effects** and **bomb casings**.

You need to start from the **first bomb effect** and try to mix it with the **last bomb casing**. If the **sum** of their values is **equal** to **any of the materials in the table below** – **create the bomb corresponding** to the **value** and **remove both** bomb materials. Otherwise, just **decrease** the value of the **bomb casing by 5**. You need to **stop** combining when you have **no more bomb effects** or **bomb casings,** or you successfully filled the bombs pouch.

Bombs:

- **Datura Bombs: 40**
- **Cherry Bombs: 60**
- **Smoke Decoy Bombs: 120**

To fill the bomb pouch, Ezio needs **three of each** of the **bomb types**.

Input

- On the **first line**, you will receive the integers representing the **bomb effects**, **separated** by **", "**.
- On the **second line**, you will receive the integers representing the **bomb casings**, **separated** by ", ".

Output

- On the **first** line, print:
  - if Ezio **succeeded** to fulfill the bomb pouch: **`"Bene! You have successfully filled the bomb pouch!"`**
  - if Ezio **didn't succeed** to fulfill the bomb pouch: **`"You don't have enough materials to fill the bomb pouch."`**
- On the **second** line, print all bomb effects left:
  - If there are no bomb effects: **`"Bomb Effects: empty"`**
  - If there are effects: **`"Bomb Effects: {bombEffect1}, {bombEffect2}, (…)"`**
- On the **third** line, print all bomb casings left:
  - If there are no bomb casings: **`"Bomb Casings: empty"`**
  - If there are casings: **`"Bomb Casings: {bombCasing1}, {bombCasing2}, (…)"`**
- Then**,** you need to print **all** bombs and the **count you have of them**, ordered **alphabetically**:
  - **`"Cherry Bombs: {count}"`**
  - **`"Datura Bombs: {count}"`**
  - **`"Smoke Decoy Bombs: {count}"`**

Constraints

- All of the given numbers will be valid integers in the range **[0, 120]**.
- There will be no cases with negative material.

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Examples

Input	Output
5, 25, 25, 115 5, 15, 25, 35	You don't have enough materials to fill the bomb pouch. Bomb Effects: empty Bomb Casings: empty Cherry Bombs: 0 Datura Bombs: 3 Smoke Decoy Bombs: 1
Comment	
1) 5 + 35 = 40 -> Datura Bomb. Remove both. 2) 25 + 25 = 50 -> can't create bomb. Bomb casing should be decreased with 5 -> 20 3) 25 + 20 = 45 -> can't create bomb. Bomb casing should be decreased with 5 -> 15 4) 25 + 15 = 40 -> Datura Bomb. Remove both …	

Input	Output
30, 40, 5, 55, 50, 100, 110, 35, 40, 35, 100, 80 20, 25, 20, 5, 20, 20, 70, 5, 35, 0, 10	Bene! You have successfully filled the bomb pouch! Bomb Effects: 100, 80 Bomb Casings: 20 Cherry Bombs: 3 Datura Bombs: 4 Smoke Decoy Bombs: 3
Comment	
… After creating a bomb with bomb effect 35 and bomb casing 25, have created 3 Cherry bombs, 4 Datura bombs, and 3 Smoke Decoy bombs. From all of the bomb types we have 3 bombs, so the program ends.	

"Nothing is true; everything is permitted"

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Snake

Everybody remembers the old snake game. Now it is time to create your own.

You will be given an integer **n** for the **size** of the snake territory with **square** shape. On the next **n** lines, you will receive the **rows** of the territory. The snake will be placed on a **random position**, marked with the letter **'S'**. On random positions there will be food, marked with **'*'**. There might also be a **lair** on the territory. The lair has two burrows. They are **marked** with the **letter** - **'B'**. **All of the empty positions** will be marked with **'-'**.

Each turn, you will be given **command** for the **snake's movement**. When the snake moves it leaves a trail marked with **'.'**

Move commands will be: **"up"**, "**down**", "**left**", "**right**".

If the snake **moves** to a **food**, it eats the food and increases the food quantity with one.

If it goes inside of a **burrow**, it **goes out** on the **position** of the **other burrow** and then **both** burrows **disappear**. If the snake **goes out** of its territory, it loses, can't return back and the program ends. The snake needs **at least 10 food quantity** to win.

When **the snake** has gone outside of its territory **or has eaten enough food,** the game **ends**.

Input

- On the first line, you are given the integer **n** – the size of the **square** matrix.
- The **next n lines** holds the values for every **row**.
- On each of the next lines you will get a move command.

Output

- On the first line:
  - If the snake goes out of its territory, print: **"Game over!"**
  - If the snake eat enough food, print: **"You won! You fed the snake."**
- On the second line print all food eaten: **"Food eaten: {food quantity}"**
- In the end print the matrix.

Constraints

- The size of the **square** matrix will be between **[2…10].**
- There will **always** be **0** or **2** burrows, marked with - **'B'**.
- The snake position will be marked with **'S'**.
- The snake will **always** either go outside its territory or eat enough food.
- There will be no case in which the snake will go through itself.

Examples

Input	Output	Comments
6 -----S ----B-	Game over! Food eaten: 1 ----..	1) left ----S. ----B- 2) down ----.. ----.- 3) down ----.. ----.- 5) down ----.. ----.-

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

------ ------ --B--- --*--- left down down down left	----.- ------ ------ --.--- --.---	------ ------ ------ ------ ------ ------ ------ ------ --B--- --S--- --.--- --.--- --*--- --*--- --S--- --.--- 3) eat the food: '*' (5, 2) 5) the snake goes out from its territory and the program ends
7 --***S- --*---- --***-- ---**-- ---*--- ---*--- ---*--- left left left down down right right down left down	You won! You fed the snake. Food eaten: 10 --....- --.---- --...-- ---..-- ---S--- ---*--- ---*---	

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

List Manipulator

Write a function called **`list_manipulator`** which receives a **list of numbers** as **first parameter** and **different amount** of other parameters. The **second** parameter might be **`"add"`** or **`"remove"`**. The **third** parameter might be **`"beginning"`** or **`"end"`**. There **might** or **might not** be any **other parameters** (numbers):

- In case of **`"add"`** and **`"beginning"`**, **add** the given **numbers** to the **beginning** of the given **list of numbers** and **return the new list**
- In case of **`"add"`** and **`"end"`**, **add** the given **numbers** to the **end** of the given **list of numbers** and **return the new list**
- In case of **`"remove"`** and **`"beginning"`**
    - If there is **another parameter** (number), **remove** that **amount** of numbers from the **beginning** of the **list of numbers**.
    - If there are **no other parameters**, **remove** only the **first element** of the list.
    - Finaly, **return the new list**
- In case of **`"remove"`** and **`"end"`**
    - If there is **another parameter** (number), **remove** that **amount** of numbers from the **end** of the **list of numbers**.
    - Otherwise if there are **no other parameters**, **remove** only the **last element** of the list.
    - Finaly, **return the new list**

For more clarifications, see the examples below.

Input

- There will be **no input**
- **Parameters** will be passed to your function

Output

- The function should **return the new list of numbers**

Examples

Test Code	Output
`print(list_manipulator([1,2,3], "remove", "end"))`	`[1, 2]`
`print(list_manipulator([1,2,3], "remove", "beginning"))`	`[2, 3]`
`print(list_manipulator([1,2,3], "add", "beginning", 20))`	`[20, 1, 2, 3]`
`print(list_manipulator([1,2,3], "add", "end", 30))`	`[1, 2, 3, 30]`
`print(list_manipulator([1,2,3], "remove", "end", 2))`	`[1]`
`print(list_manipulator([1,2,3], "remove", "beginning", 2))`	`[3]`
`print(list_manipulator([1,2,3], "add", "beginning", 20, 30, 40))`	`[20, 30, 40, 1, 2, 3]`
`print(list_manipulator([1,2,3], "add", "end", 30, 40, 50))`	`[1, 2, 3, 30, 40, 50]`

SoftUni

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Follow us: