# Mid Exam Preparation – 14 October 2024

## Counter Strike

**Link**: <https://alpha.judge.softuni.org/contests/03-programming-fundamentals-mid-exam-retake/2305/practice#0>

Write a program that **keeps track of every won** battle against an **enemy**. You will receive **initial energy**. Afterward, you will start receiving the **distance** you need **to reach an enemy** until the **"End of battle"** command is given, or you **run out of energy**.

The **energy** you need for reaching an enemy is **equal to the distance you receive**. Each time you reach an enemy, you **win** a battle, and your **energy is reduced**. Otherwise, if you don't have **enough energy** to reach an enemy, **end the program** and **print**: **"Not enough energy! Game ends with {count} won battles and {energy} energy"**.

Every **third won battle** increases **your energy with the value of your current count of won battles**.

Upon receiving the **"End of battle"** command**,** print the **count of won battles** in the following format:

"Won battles: {count}. Energy left: {energy}"

### Input / Constraints

* On the **first line,** you will receive **initial energy** – an **integer [1-10000]**.
* On the **following lines,** you will be receiving the **distance** of an enemy – an **integer** **[1-10000]**

### Output

* The description contains the proper output messages for each case and the format they should be printed.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 100  10  10  10  1  2  3  73  10 | Not enough energy! Game ends with 7 won battles and 0 energy | The initial energy is 100. The first distance is 10, so we subtract 10 from 100, and we consider this a **won** battle. We are left with 90 energy. Next distance – 10, and 80 energy left.  Next distance – 10, 3 won battles and 70 energy, but since we have 3 won battles, we increase the energy with the current count of won battles, in this case – **3, and it becomes 73**.  The last distance we receive – **10** is unreachable since we have **0** energy, so we print the appropriate message, and the program ends. |
| 200  54  14  28  13  End of battle | | Won battles: 4. Energy left: 94 |  |

## Shopping List

**Link**: <https://alpha.judge.softuni.org/contests/04-programming-fundamentals-mid-exam/2031/practice#1>

*It’s the end of the week and it is time for you to go shopping, so you need to create a shopping list first.*

### Input

You will receive an **initial list** with groceries separated by **"!"**.

After that you will be receiving **4 types** of commands, until you receive **"Go Shopping!"**

* **Urgent {item} -** **add** the item at the **start** of the list. If the item **already exists,** skip this command.
* **Unnecessary {item} - remove** the item with the given name, only **if it exists** in the list. Otherwise skip this command.
* **Correct {oldItem} {newItem} –** if the item with the given **old name** exists, **change** its name with the **new** one. If it **doesn't exist**, skip this command.
* **Rearrange {item} -** if the grocery exists in the list, **remove** it from its **current position** and **add** it at the **end** of the list.

### Constraints

* There won`t be any duplicate items in the initial list

### Output

Print the **list** with all the groceries, joined by **", ".**

* **"{firstGrocery}, {secondGrocery}, …{nthGrocery}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Tomatoes!Potatoes!Bread  Unnecessary Milk  Urgent Tomatoes  Go Shopping! | Tomatoes, Potatoes, Bread |
| Milk!Pepper!Salt!Water!Banana  Urgent Salt  Unnecessary Grapes  Correct Pepper Onion  Rearrange Grapes  Correct Tomatoes Potatoes  Go Shopping! | Milk, Onion, Salt, Water, Banana |

## Memory Game

**Link:** <https://alpha.judge.softuni.org/contests/01-programming-fundamentals-mid-exam-retake/2517/practice#12>

Write a program that recreates the **Memory game**.

On the first line, you will **receive a sequence of elements**. Each element in the sequence **will have a** **twin**. Until the player receives **"end"** from the console, you will receive **strings with two integers** separated by a space, representing **the indexes** of elements in the sequence.

If the player **tries to cheat** and enters **two equal indexes** or indexes which are **out of bounds of the sequence**, you should **add** two matching elements at the middle of the sequence in the following format:

**"-{number of moves until now}a"**

Then print this message on the console:

**"Invalid input! Adding additional elements to the board"**

### Input

* On the **first** line**,** you will receive a **sequence of elements.**
* On the **following** lines, you will receive **integers** until the command **"end".**

### Output

* Every time the player hit **two matching elements**, you should **remove** them from the sequence and **print** on the console the following message:

**"Congrats! You have found matching elements - ${element}!"**

* If the player hit **two different elements**, you should **print** on the console the following message:

**"Try again!"**

* If the player hit **all matching elements** before he receives **"end"** from the console, you should **print** on the console the following message:

**"You have won in {number of moves until now} turns!"**

* If the player receives **"end"** **before** **he hits all matching elements**, you should **print** on the console the following message:

**"Sorry you lose :(**

**{the current sequence's state}"**

### Constraints

* **All elements in the sequence will always have a matching element.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 1 2 2 3 3 4 4 5 5  1 0  -1 0  1 0  1 0  1 0  end | Congrats! You have found matching elements - 1!  Invalid input! Adding additional elements to the board  Congrats! You have found matching elements - 2!  Congrats! You have found matching elements - 3!  Congrats! You have found matching elements - -2a!  Sorry you lose :(  4 4 5 5 |
| **Comment** | |
| 1)  1 0  1 1 2 2 3 3 4 4 5 5 –> 1 = 1, equal elements, so remove them. Moves: 1  2)  -1 0  -1 is invalid index so we add additional elements  2 2 3 3 -2а -2а 4 4 5 5, Moves: 2  3)  1 0  2 2 3 3 -2а -2а 4 4 5 5 -> 2 = 2, equal elements, so remove them. Moves: 3  4)  1 0  3 3 -2а -2а 4 4 5 5 -> 3 = 3, equal elements, so remove them. Moves: 4  5)  1 0  -2а -2а 4 4 5 5 -> -2а = -2а, equal elements, so remove them. Moves: 5  6)  You receive the end command.  There are still elements in the sequence, so the player loses the game.  Final state - 4 4 5 5 | |
| a 2 4 a 2 4  0 3  0 2  0 1  0 1  end | Congrats! You have found matching elements - a!  Congrats! You have found matching elements - 2!  Congrats! You have found matching elements - 4!  You have won in 3 turns! |
| a 2 4 a 2 4  4 0  0 2  0 1  0 1  end | Try again!  Try again!  Try again!  Try again!  Sorry you lose :(  a 2 4 a 2 4 |