# Programming Fundamentals Mid Exam Retake 07 April 2020

## Problem 1. Counter Strike

Write a program that **keeps track of every won** battle against an **enemy**. You will receive **initial energy**. Afterwards you will start receiving the **distance** you need to **go to reach an enemy** until the **"End of battle"** command is given, or until 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, your **energy is reduced.** This is considered a successful battle (**win**). If you don't have **enough energy** to reach an the enemy, print:

**"Not enough energy! Game ends with {count} won battles and {energy} energy"**

and **end the program.**

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 **next lines,** you will be receiving **distance** of the enemy – an **integer** **[1-10000]**

### Output

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

### 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 | 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 battle, in this case – **3 and it becomes 73**.  The last distance we receive – **10** is unreachalble 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 |  |

## Problem 2. Shoot for the Win

Write a program that helps you keep track of your **shot targets**. You will receive a **sequence with integers**, separated by single space, representing targets and their value. Afterwards, you will be receiving indices until the **"End"** command is given and you need to print the **targets** and the **count of shot targets**.

Every time you receive an **index**, you need to shoot the target on that index, **if it is possiblie**.

Everytime you **shoot a target**, its value becomes **-1 and it is considered shot**. Along with that you also need to:

* **Reduce** all the other **targets**, which have **greater values** than your **current** target, **with its value**.
* All the **targets**, which **have less than or equal** value to the **shot target**, you need to **increase** **with its value.**

**Keep in mind that you can't shoot a target, which is already shot.** **You also can't increase or reduce a target, which is considered shot.**

When you receive the **"End"** command, print the targets in their current state and the **count of shot targets** in the following format:

**"Shot targets: {count} -> {target1} {target2}… {targetn}"**

### Input / Constraints

* On the **first line** of input you will receive a **sequence** of **integers**, **separated** by **a single space – the targets sequence**.
* On the **next lines**, until the **"End"** command you be receiving **integers** each on a single line – **the index of the target to be shot.**

### Output

* The format of the output is described above in the problem description.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 24 50 36 70  0  4  3  1  End | Shot targets 3 -> -1 -1 130 -1 | First we shoot target on index 0. It becomes equal to -1 and we start going through the rest of the targets. Since 50 is more than 24, we reduce it to 26 and 36 to 12 and 70 to 46. The sequence looks like that:  **-1 26 12 46**  The next index is invalid, so we don't do anything. Index 3 is valid and after the operations our sequence should look like that:  **-1 72 58 -1**  Then we take the first index with value 72 and our sequence looks like that:  **-1 -1 130 -1**  Then we print the result after the **"End"** command. |
| 30 30 12 60 54 66  5  2  4  0  End | Shot targets: 4 -> -1 120 -1 66 -1 -1 |  |

## Problem 3. Moving Target

You are at the shooting gallery again and you need a program that helps you keep track of moving targets. On the first line, you will receive a **sequence of targets with their integer values**, split by a **single space**. Then, you will start receiving **commands for manipulating the targets**, until the **"End"** command. The commands are the following:

* **Shoot {index} {power}**
  + Shoot the target at the index, **if it exists** by **reducing** its **value** by the **given** **power** (**integer value**).A target is considered **shot** when **its value reaches 0**.
  + Remove the target, **if it is shot**.
* **Add {index} {value}**
  + Insert a target with the received value at the received **index, if it exist**. If not, print: **"Invalid placement!"**
* **Strike {index} {radius}**
  + Remove the **target at the given index** and the **ones before and after it depending on the radius, if such exist. If any of the indices in the range is invalid print:**

**"Strike missed!" and skip this command.**

**Example:** **Strike 2 2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | {radius} | {radius} | {strikeIndex} | {radius} | {radius} |  |  |

* **End**
  + Print the sequence with targets in the following format:

**{target1}|{target2}…|{targetn}**

### Input / Constraints

* On the **first line** you will receive **the sequence of targets** – **integer values [1-10000]**.
* On the **next lines,** until the **"End"** will be receiving the command described above – **strings**.
* There will never be a case when **"Strike"** command would empty the whole sequence.

### Output

* Print the appropriate message in case of **"Strike"** command if necessary.
* In the end, print the sequence of targets in the format described above.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 52 74 23 44 96 110  Shoot 5 10  Shoot 1 80  Strike 2 1  Add 22 3  End | Invalid placement!  52|100 | The first command is "**Shoot**", so we reduce the target on **index** **5**, which is valid, with the given **power** – **10**.  Then we receive the same command but we need to reduce the target on the 1st index, with power 80. The value of this target is 74, so it is considered shot and we **remove** it.  Then we receive the "**Strike**" command on the 2nd index and we need to check if the range with radius 1 is valid:  **52 23 44 96 100**  And it is, so we **remove** the targets.  At last we receive the "**Add**" command, but the index is **invalid** so we print the appropriate **message** and in the end we have the following result:  **52|100** |
| 47 55 85 78 99 20  Shoot 1 55  Shoot 8 15  Strike 2 3  Add 0 22  Add 2 40  Add 2 50  End | Strike missed!  22|47|50|40|85|78|99|20 |  |