# Problem 1 – Guinea Pig

A problem for exam preparation for the [Programming Fundamentals Course @ SoftUni](https://softuni.bg/trainings/3729/programming-fundamentals-with-csharp-may-2022)

Submit your solutions in the SoftUni Judge system [here](https://judge.softuni.org/Contests/Practice/Index/2031#0)

*Merry has a guinea pig named Puppy, that she loves very much. Every month she goes to the nearest pet store and buys him everything he needs – food, hay and cover.*

On the **first three lines**, you will receive **the quantity of food**, **hay** and **cover**, which Merry buys for a **month (30 days)**. On the **fourth line**, you will receive the **guinea pig's weight**.

**Every day** Puppy eats **300 gr of food**.

**Every** **second** day Merry **first feeds the pet**, then gives it a **certain amount of hay** **equal to** **5%** of the rest of the **food**. On **every** **third** day, Merry puts Puppy **cover** with **a quantity of** **1/3** of its **weight**.

**Calculate** whether the quantity of **food, hay and cover** will be enough for a **month**.

**If Merry runs out of food, hay, or cover, stop the program!**

## Input

* **On the first line – quantity food in kilograms** - afloating-point number in the range **[0.0…10000.0].**
* **On the second line – quantity hay in kilograms** - afloating-point number in the range **[0.0…10000.0].**
* **On the third line – quantity cover in kilograms** - afloating-point number in the range **[0.0…10000.0].**
* **On the fourth line – guinea's weight in kilograms** - afloating-point number in the range **[0.0…10000.0].**

## Output

* If the food, the hay and the cover are enough, print:
  + **"Everything is fine! Puppy is happy! Food: {excessFood}, Hay: {excessHay}, Cover: {excessCover}."**
* If one of the things is not enough, print:
  + **"Merry must go to the pet store!"**

**The output values must be formatted to the second decimal place!**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10  5  5.2  1 | Everything is fine! Puppy is happy! Food: 1.00, Hay: 1.10, Cover: 1.87. |
| You receive food – **10000**, hay – **5000**, cover – **5200**, weight – **1000** (in grams).  On the first day, Merry gives Puppy 300gr food – 9700gr food left.  On the second day, the food left is **9400gr**, so the needed hay is **9400 \* 5% = 470**,and thehay left is **4530.**  On the third day, the cover left is **4866.67,** and the food left is **9100**,and so on.  On the last day, Merry has: food – 1.00, hay – 1.10, and cover – 1.87. | |
| 1  1.5  3  1.5 | Merry must go to the pet store! |
| 9  5  5.2  1 | Merry must go to the pet store! |

# Problem 2 – The Lift

A problem for exam preparation for the [Programming Fundamentals Course @ SoftUni](https://softuni.bg/trainings/3729/programming-fundamentals-with-csharp-may-2022)

Submit your solutions in the SoftUni Judge system [here](https://judge.softuni.org/Contests/Practice/Index/2517#1)

Write a program that **finds a place for the tourist on a lift.**

Every wagon should have **a maximum of 4 people on it**. If a wagon is full, you should direct the people to **the next one with space** available.

### Input

* **On the first line,** you will receive **how many people** are waiting to get **on the lift.**
* **On the second line**, you will receive the **current state of the lift separated by a single space** **' '**.

### Output

**When there is no more available space left on the lift**, or there are **no more people in the queue**, you should print on the console the final state of the lift's wagons separated by **' '** and one of the following messages:

* If there are no more people and the lift has empty spots, you should print:

**"The lift has empty spots!**

**{wagons separated by ' '}"**

* If there are still people in the queue and no more available space, you should print:

**"There isn't enough space! {people} people in a queue!**

**{wagons separated by ' '}"**

* If the lift is full and there are no more people in the queue, you should print only the wagons separated by **" "**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 15  0 0 0 0 | The lift has empty spots!  4 4 4 3 |
| **Comment** | |
| First state - 4 0 0 0 -> 11 people left  Second state – 4 4 0 0 -> 7 people left  Third state – 4 4 4 0 -> 3 people left | |
| **Input** | **Output** |
| 20  0 2 0 | There isn't enough space! 10 people in a queue!  4 4 4 |
| **Comment** | |
| First state - 4 2 0 -> 16 people left  Second state – 4 4 0 -> 14 people left  Third state – 4 4 4 -> 10 people left, but there're no more wagons. | |

# Problem 3 – Moving Target

A problem for exam preparation for the [Programming Fundamentals Course @ SoftUni](https://softuni.bg/trainings/3729/programming-fundamentals-with-csharp-may-2022)

Submit your solutions in the SoftUni Judge system [here](https://judge.softuni.org/Contests/Practice/Index/2305#2)

You are at a shooting gallery 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**).
  + Remove the target, **if it is shot**. A target is considered **shot** when **its value reaches 0**.
* **"Add {index} {value}"**
  + Insert a target with the received value at the received **index, if it exists**.
  + 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 **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 and **end** **the program**:

**"{target1}|{target2}…|{targetn}"**

### Input / Constraints

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

### Output

* Print the appropriate message in case of any 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** |
| 1 2 3 4 5  Strike 0 1  End | Strike missed!  1|2|3|4|5 |  |