# Problem 1 – Splinter Trip

**Sam the Spy** just got his shiny new military aircraft, the **C-147B Paladin**, and a shiny new mission to carry out. The problem is, he needs to fly there using the Paladin, and you're going to help him with **calculating the fuel consumption** and **total flight time**.

The Paladin, being a big plane, **consumes** a lot of **fuel** - **25L per mile** to be exact.

Also, before taking off, the commanding Fuel Consumption Officer (you) needs to calculate the **miles traveled in heavy winds**. **Heavy** **winds** need **1.5 times more fuel**.

Finally, since fuel consumption is always going to **vary** a little, we need to have a **bit more fuel** just in case. So, the **total fuel amount** we put in needs to **increase by 5%**.

When we calculate the fuel consumption, we need to print it on the console in the following format:

* “Fuel needed: {totalFuelNeeded}L”

After all of these calculations, we need to see if the **fuel in the tank** will be **enough**:

* If it’s enough, print:
  + “Enough with {remainingFuel}L to spare!”
* If the fuel won’t be enough, print:
  + “We need {fuelNeeded}L more fuel.”

All **floating-point** numbers in the output are **rounded to the second decimal place**.

### Input

* First line – the **trip distance** in **miles** – **floating-point number** in **range [1.00…250000.00]**.
* Second line – the **fuel tank capacity** in **liters** – **floating-point number** in **range [1.00…100000.00]**.
* Third line – the **miles spent in heavy winds** – **floating-point number** in **range [0.00…50000.00]**.

### Output

* First line – The **total fuel consumption** – **rounded to the second decimal place**
* Second line – whether the plane will have **enough fuel**

All the output needs to be as per the formats stated **above**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 500.5  14000  50 | Fuel needed: 13794.38L  Enough with 205.63L to spare! | Travel distance – **500.5** miles  Fuel tank capacity – **14000** liters  Miles in heavy winds – **50**  Miles in non-heavy winds – **500.5-50** 🡺 **450.5**  Non-heavy winds consumption – **450.5\*25** 🡺 **11262.5** liters  Heavy winds consumption – **50\*(25\*1.5)** 🡺 **1875** liters  Fuel consumption 🡺 **11262.5+1875** 🡺 **13137.5** liters  Tolerance – **13137.5\*5%** 🡺 **656.875** liters  Total Fuel Consumption 🡺 **13137.5+656.875** 🡺 **13794.375** liters  Remaining fuel – **14000-13794.375** 🡺 **205.625** liters (enough) |
| 9000  235000  230 | Fuel needed: 239268.75L  We need 4268.75L more fuel. | Travel distance – **9000** miles  Fuel tank capacity – **235000** liters  Miles in heavy winds – **230**  Miles in non-heavy winds – **9000-230** 🡺 **8770**  Non-heavy winds consumption – **8770\*25** 🡺 **219250** liters  Heavy winds consumption – **230\*(25\*1.5)** 🡺 **8625** liters  Fuel consumption 🡺 **219250+8625** 🡺 **227875** liters  Tolerance – **227875\*5%** 🡺 **11393.750** liters  Total Fuel Consumption 🡺 **227875+11393.750** 🡺 **239268.750** liters  Remaining fuel – **235000-239268.750** 🡺 -**4268.750** liters (not enough) |
| 1000  26250  0 | Fuel needed: 26250.00L  Enough with 0.00L to spare! | Travel distance – **1000** miles  Fuel tank capacity – **26250** liters  Miles in non-heavy winds – **1000**  Non-heavy winds consumption – **1000\*25** 🡺 **25000** liters  Fuel consumption – **25000** liters  Tolerance – **25000\*5%** 🡺 **1250** liters  Total Fuel Consumption 🡺 **25000+1250** 🡺 **26250** liters  Remaining fuel – **26250-26250** 🡺 **0** liters (enough) |

# Problem 2 – SpyGram

After arriving from the trip from the Splinter Trip problem, our hero **Sam** is feeling a bit jet lagged, but he’s ready to go to work! He needs to receive **orders** from his **commanding officers** through the sophisticated messaging app, called **SpyGram**. The app uses a sophisticated algorithm to **encrypt** messages. Since you’re tired of jumbling text by hand for 8 hours a day, you decide to write an **algorithm** to do it for you.

**Sending** messages works the following way – an **outgoing** message **must** follow the following **format**:

* TO: {recipient}; MESSAGE: {message};

Here’s what an example message looks like: “TO: GOSHO; MESSAGE: hi.;”. If any message is **not** in this format, you should **ignore it**.

Both parties are given a **private key**, which consists of **digits** of a **variable length**, with which to **encrypt**/**decrypt** messages. The encryption algorithm is simple:

First, we take the message in the format above, then we **shift right** the **first character** of our message by the **value** of the **first character** of our **private key** in the **ASCII table,** the **second character** by the **second private key character,** and so on. If we **run out** **of characters** in our **private key** (such as when our message is longer than our private key), then we **start over** from the beginning of our **private key**.

So, with an **example message** of “**hello”**, and an **example private key** of **123**, it would look like this:

hello -> igomq

* h gets shifted right by **1 character**
* e gets shifted right by **2 characters**
* l gets shifted right by **3 characters**
* l gets shifted right by **1 character** (we **ran out** of characters in the **private key** and **rolled over** to the start)
* o gets shifted right by **2 characters**.

### Input

The **first line** of our input consists of our **private key**.

Until we receive the command “**END**”, we’ll start receiving **non-encrypted messages to send** to the command center. If we receive a **message to** **send**, we need to put it in our **pending messages** collection.

After that, we need to **sort** the sent messages by **sender name** in **ascending order**.

### Output

To send a message, all we have to do is **encrypt it** and **print** it on the console in the standard **outgoing message** format.

### Constraints

* All **valid** recipient/sender names will be **UPPERCASE** and contain only **Latin letters**
* Messages can contain **ANY** **ASCII character**

### Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 13234  TO: GRIM; MESSAGE: hello;  TO: ARCHER; MESSAGE: sneak around it;  END | UR<#ESFJHV<#OHWTDIH>!vphel#cusvqf#mu>  UR<#KSLO>$NHUVEHH<#lfonr? |
| 142325555  TO: KOBIN; MESSAGE: one two three;  TO: KESTREL; MESSAGE: affirmative;  T: REGAN; MESSAGE: help me;  TO: TOMCLANCY; MESSAGE: let's get to work;  TO: kestrel; MESSAGE: affirmative;  END | US<#MJXYWFP=#OJXXFHI<#ckknwnevlxj@  US<#MTGNS<$OHUXFLJ;$qqg%y|t!xjugj@  US<#VTRHQBRE\=%RJXTEIH<%qjy(w"jgy%yt!{qum@ |
| 82738  TO: ARCHER; MESSAGE: affirmative;  FROM: SAM; MESSAGE: i'm pinned down;  TO: SAM; MESSAGE: 55% done;  FROM: SAM; MESSAGE: infiltrate the storage facility;  END | \QA#IZEOHZC"TH[[CNHB(cmiqzohwq~gB  \QA#[IOB#UMUZDOM<'8=-"krvm= |

# Problem 3. Spyfer

Sam the spy has created a new system for tracking called “Spyfer”. But he needs you to write the software for it, because he is incompetent in programming.

You will receive a **sequence** of **integers**, on a **single input line**, **separated** by **spaces**. Those integers will represent coordinates.

Your task is to **check every integer**, if it is **equal** to the **sum** of its **DIRECT neighboring elements**. When you find an integer that follows this rule you should **REMOVE** the **neighboring elements**, and **REPEAT the process** again,from **the start**.

If you **reach** the **end** of **the sequence**, you should **terminate** the program and print **what’s left** of the numbers, **separated** by **spaces**.

### Input

* The input comes in the form of a **single input line**, containing the **integers**, **separated** by **space**.

### Output

* As output, you must print, what’s left of the sequence, after you’ve processed it.
* Print the elements, **separated** by **spaces**.

### Constrains

* The **integers** in the input will be in **range [0, 1000]**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 1 | 2 | We find **the first element** that is **equal** to **the sum** of its **neighboring elements**. That is **2**.  So we **remove** the **neighbours** and **return** to the **start** of the **sequence**.  There is **only 1 element**, so we just pass it and reach the end of the sequence. We print what’s left of it. |
| 3 2 8 0 1 1 4 3 6 2 | 3 8 | 1st step - 3 2 8 0 1 1 4 3 6 2  2nd step - 3 2 8 1 4 3 6 2  3rd step - 3 2 8 4 6 2  4th step - 3 2 8 6  5th step - 3 8  We reach the end of the sequence and we print what’s left of it. |

# Problem 4. NSA

Most people have probably heard the news about some dummy named Edward Snowden, who released some info about an American association called the NSA, which, by his words, was spying on innocent citizens. Who cares anyways, it’s America, not my country. Yeah well … Guess again, NSA is Global.

The NSA creates a registry of spies. The input of data comes in the following format:

{countryName} -> {spyName} -> {daysInService}

The countryName and the spyName are both **strings** which may contain **alphanumeric characters**.  
The daysInService is an **integer**.

If you get an **existent** country, **add** the **new spy** to it. If **even** the **spy** exists, **update** its daysInService, with the **new given value**.

When you get the command “quit”, you terminate the program. Then you must print all countries and their spies.

The **countries** must be **ordered** by **amount of spies** they have in **descending order**.  
The **spies** must be **ordered** by **days in service**, in **descending order**.

### Input

* The input comes in the format specified above.
* The input sequence ends when you receive the “quit” command.

### Output

* The countries and their spies must be printed in the following format:

“Country: {country1Name}  
 \*\*{spy1Name} : {spy1Score}  
 \*\*{spy2Name} : {spy2Score}  
 . . .   
 Country: {country2Name}  
 . . .”

### Constrains

* The **countryName** and the **spyName** are both **strings** which may contain **alphanumeric characters**.
* The **daysInService** will be a valid **integer** in **range [0, 231 – 1]**.
* There will be **NO** invalid input lines.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Germany -> Duffy -> 1  Australia -> Bond -> 7  America -> Bond –> 5  Germany -> Alex -> 4  America -> Donald -> 4  Germany -> Jeffrey -> 3  Australia -> Jeffrey -> 4  quit | Country: Germany  \*\*Alex : 4  \*\*Jeffrey : 3  \*\*Duffy : 1  Country: Australia  \*\*Bond : 7  \*\*Jeffrey : 4  Country: America  \*\*Bond : 5  \*\*Donald : 4 |