Exercises: Sets and Dictionaries

This document defines the **exercise assignments** for the "CSharp Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

I. Basic Sets Operations

Problem 1. Unique Usernames

Write a simple program that reads from the console a sequence of usernames and keeps a collection with only the unique ones. Print the collection on the console:

Input	Output
6 Ivan Ivan Ivan SemoMastikata Ivan Hubaviq1234	Ivan SemoMastikata Hubaviq1234

Problem 2. Sets of Elements

On the first line you are given the length of two sets n and m. On the next n + m lines there are n numbers that are in the first set and m numbers that are in the second one. Find all non-repeating element that appears in both of them, and print them at the console:

Set with length n = 4: {1, **3**, **5**, 7}

Set with length m = 3: {**3**, 4, **5**}

Set that contains all repeating elements -> {**3**, **5**}

Input	Output
4 3 1 3 5 7 3 4 5	3 5
2 2 1 3 1 5	1

Problem 3. Periodic Table

You are given an n number of chemical compounds. You need to keep track of all chemical elements used in the compounds and at the end print all unique ones in ascending order:

Input	Output
4 Ce O Mo O Ce Ee Mo	Ce Ee Mo O
3 Ge Ch O Ne Nb Mo Tc O Ne	Ch Ge Mo Nb Ne O Tc

II. Basic Dictionaries Operations

Problem 4. Count Symbols

Write a program that reads some text from the console and counts the occurrences of each character in it. Print the results in **alphabetical** (lexicographical) order.

Examples:

Input	Output		Input	Output
SoftUni rocks	: 1 time/s S: 1 time/s U: 1 time/s c: 1 time/s f: 1 time/s i: 1 time/s k: 1 time/s n: 1 time/s o: 2 time/s r: 1 time/s s: 1 time/s t: 1 time/s		Did you know Math.Round rounds to the nearest even integer?	: 9 time/s .: 1 time/s ?: 1 time/s D: 1 time/s M: 1 time/s R: 1 time/s a: 2 time/s d: 3 time/s e: 7 time/s g: 1 time/s h: 2 time/s i: 2 time/s k: 1 time/s n: 6 time/s o: 5 time/s r: 3 time/s s: 2 time/s t: 5 time/s u: 3 time/s v: 1 time/s w: 1 time/s y: 1 time/s

Problem 5. Phonebook

Write a program that receives some info from the console about **people** and their **phone numbers**.

You are free to choose the manner in which the data is entered; each **entry** should have just **one name** and **one number** (both of them strings). If you receive a name that **already exists** in the phonebook, simply update its number.

After filling this simple phonebook, upon receiving the **command** "**search**", and the **command "stop"**, your program should be able to perform a search of a contact by name and print her details in format "**{name} -> {number}**". In case the contact isn't found, print "**Contact {name} does not exist.**" Examples:

Input	Output
Nakov-0888080808 **search** Mariika Nakov **stop**	Contact Mariika does not exist. Nakov -> 0888080808
Nakov-+359888001122 RoYaL(Ivan)-666 Gero-5559393 Simo-02/987665544 **search** Simo simo RoYaL RoYaL(Ivan) **stop**	Simo -> 02/987665544 Contact simo does not exist. Contact RoYaL does not exist. RoYaL(Ivan) -> 666

Problem 6. A miner task

You are given a sequence of strings, each on a new line. Every odd line on the console is representing a resource (e.g. Gold, Silver, Copper, and so on) , and every even – quantity. Your task is to collect the resources and print them each on a new line, **unitll you receive "stop" command.**

Print the resources and their quantities in format:

{resource} –> {quantity}

The quantities inputs will be in the range [1 … 2 000 000 000]

Examples:

Input	Output
Gold 155 Silver 10 Copper 17	Gold -> 155 Silver -> 10 Copper -> 17

stop	

Problem 7. Fix emails

You are given a sequence of strings, each on a new line, **unitll you receive "stop" command**. First string is a name of a person. On the second line you receive his email. Your task is to collect their names and emails, and remove emails whose domain ends with "us" or "uk" (case insensitive). Print:

{name} – > {email}

Examples :

Input	Output
Ivan ivanivan@abv.bg Petar Ivanov petartudjarov@abv.bg Mike Tyson myke@gmail.us **stop**	Ivan -> ivanivan@abv.bg Petar Ivanov -> petartudjarov@abv.bg

Problem 8. Hands of cards

You are given a sequence of people and for every person what cards he draws from the deck. The input will be separate lines in the format:

{personName}: {PT, PT, PT,… PT}

Where P (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A) is the power of the card and T (S, H, D, C) is the type. The input ends when a **"JOKER"** is drawn. The name can contain any ASCII symbol except **':'**. The input will always be valid and in the format described, there is no need to check it.

A single person cannot have more than one card with the same power and type, if he draws such a card he discards it. The people are playing with multiple decks. Each card has a value that is calculated by the power multiplied by the type. Powers **2 to 10** have the same value and **J to A** are **11 to 14**. Types are mapped to multipliers the following way (**S -> 4, H-> 3, D -> 2, C -> 1**).

Finally print out the total value each player has in his hand in the format:

{personName}: {value}

Examples:

Input	Output
Pesho: 2C, 4H, 9H, AS, QS Slav: 3H, 10S, JC, KD, 5S, 10S Peshoslav: QH, QC, QS, QD Slav: 6H, 7S, KC, KD, 5S, 10C Peshoslav: QH, QC, JS, JD, JC Pesho: JD, JD, JD, JD, JD, JD JOKER	Pesho: 167 Slav: 175 Peshoslav: 197

III. Sets and Dictionaries Exercises

Problem 9. * User Logs

Marian is a famous system administrator. The person to overcome the security of his servers has not yet been born. However, there is a new type of threat where users flood the server with messages and are hard to be detected since they change their IP address all the time. Well, Marian is a system administrator and is not so into programming. Therefore, he needs a skillful programmer to track the user logs of his servers. You are the chosen one to help him!

You are given an input in the format:

IP=(IP.Address) message=(A&sample&message) user=(username)

Your task is to parse the ip and the username from the input and for **every user**, you have to display **every ip** from which the corresponding user has sent a message and the **count of the messages** sent with the corresponding ip. In the output, the usernames must be **sorted alphabetically** while their IP addresses should be displayed in the **order of their first appearance.** The output should be in the following format:

```
username:
IP => count, IP => count.
```

For example, given the following input - **IP=192.23.30.40 message='Hello&derps.' user=destroyer**, you have to get the username **destroyer** and the IP **192.23.30.40** and display it in the following format:

```
destroyer:
192.23.30.40 => 1.
```

The username destroyer has sent a message from ip 192.23.30.40 once.

Check the examples below. They will further clarify the assignment.

Input

The input comes from the console as **varying number** of lines. You have to parse every command until the command that follows is **end.** The input will be in the format displayed above, there is no need to check it explicitly.

Output

For every user found, you have to display each log in the format:

```
username:
IP => count, IP => count…
```

The IP addresses must be split with a comma, and each line of IP addresses must end with a dot.

Constraints

- The number of commands will be in the range [1..50]
- The IP addresses will be in the format of either **IPv4** or **IPv6.**
- The messages will be in the format: **This&is&a&message**
- The username will be a string with length in the range [3..50]
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output
IP=192.23.30.40 message='Hello&derps.' user=destroyer IP=192.23.30.41 message='Hello&yall.' user=destroyer IP=192.23.30.40 message='Hello&hi.' user=destroyer IP=192.23.30.42 message='Hello&Dudes.' user=destroyer end	destroyer: 192.23.30.40 => 2, 192.23.30.41 => 1, 192.23.30.42 => 1.
IP=FE80:0000:0000:0000:0202:B3FF:FE1E:8329 message='Hey&son' user=mother IP=192.23.33.40 message='Hi&mom!' user=child0 IP=192.23.30.40 message='Hi&from&me&too' user=child1 IP=192.23.30.42 message='spam' user=destroyer IP=192.23.30.42 message='spam' user=destroyer IP=192.23.50.40 message='' user=yetAnotherUsername IP=192.23.50.40 message='comment' user=yetAnotherUsername IP=192.23.155.40 message='Hello.' user=unknown end	child0: 192.23.33.40 => 1. child1: 192.23.30.40 => 1. destroyer: 192.23.30.42 => 2. mother: FE80:0000:0000:0000:0202:B3FF:FE1E:8329 => 1. unknown: 192.23.155.40 => 1. yetAnotherUsername: 192.23.50.40 => 2.

Problem 10. * Population Counter

So many people! It's hard to count them all. But that's your job as a statistician. You get raw data for a given city and you need to aggregate it.

On each input line you'll be given data in format: **"city|country|population"**. There will be **no redundant whitespaces anywhere** in the input. Aggregate the data **by country and by city** and print it on the console. For each country, print its **total population** and on separate lines the data for each of its cities. **Countries should be ordered by their total population in descending order** and within each country, the **cities should be ordered by the same criterion**. If two countries/cities have the same population, keep them **in the order in which they were entered.** Check out the examples; follow the output format strictly!

Input

- The input data should be read from the console.
- It consists of a variable number of lines and ends when the command "**report**" is received.
- The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

- The output should be printed on the console.
- Print the aggregated data for each country and city in the format shown below.

Constraints

- The name of the city, country and the population count will be separated from each other by **a pipe ('|')**.
- The **number of input lines** will be in the range [2 … 50].
- A city-country pair will not be repeated.
- The **population count** of each city will be an integer in the range [0 … 2 000 000 000].

- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
Sofia|Bulgaria|1000000 report	Bulgaria (total population: 1000000) =>Sofia: 1000000

Input	Output
Sofia|Bulgaria|1 Veliko Tarnovo|Bulgaria|2 London|UK|4 Rome|Italy|3 report	UK (total population: 4) =>London: 4 Bulgaria (total population: 3) =>Veliko Tarnovo: 2 =>Sofia: 1 Italy (total population: 3) =>Rome: 3

Problem 11. * Logs Aggregator

You are given a sequence of access logs in format **<IP> <user> <duration>**. For example:

- 192.168.0.11 peter 33
- 10.10.17.33 alex 12
- 10.10.17.35 peter 30
- 10.10.17.34 peter 120
- 10.10.17.34 peter 120
- 212.50.118.81 alex 46
- 212.50.118.81 alex 4

Write a program to aggregate the logs data and print **for each user** the **total duration** of his sessions and a **list of unique IP addresses** in format "**<user>: <duration> [<IP$_1$>, <IP$_2$>, …]**". Order the **users alphabetically**. Order the **IPs alphabetically**. In our example, the output should be the following:

- alex: 62 [10.10.17.33, 212.50.118.81]
- peter: 303 [10.10.17.34, 10.10.17.35, 192.168.0.11]

Input

The input comes from the console. At the first line a number **n** stays which says how many log lines will follow. Each of the next n lines holds a log information in format **<IP> <user> <duration>**. The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print **one line for each user** (order users alphabetically). For each user print its sum of durations and all of his sessions' IPs, ordered alphabetically in format **<user>: <duration> [<IP$_1$>, <IP$_2$>, …]**. Remove any duplicated values in the IP addresses and order them alphabetically (like we order strings).

Constraints

- The **count** of the order lines **n** is in the range [1…1000].
- The **<IP>** is a standard IP address in format **a.b.c.d** where **a**, **b**, **c** and **d** are integers in the range [0…255].
- The **<user>** consists of only of **Latin characters**, with length of [1…20].
- The **<duration>** is an integer number in the range [1…1000].

- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output
7 192.168.0.11 peter 33 10.10.17.33 alex 12 10.10.17.35 peter 30 10.10.17.34 peter 120 10.10.17.34 peter 120 212.50.118.81 alex 46 212.50.118.81 alex 4	alex: 62 [10.10.17.33, 212.50.118.81] peter: 303 [10.10.17.34, 10.10.17.35, 192.168.0.11]
2 84.238.140.178 nakov 25 84.238.140.178 nakov 35	nakov: 60 [84.238.140.178]

Problem 12. * Legendary Farming

You've beaten all the content and the last thing left to accomplish is own a legendary item. However, it's a tedious process and requires quite a bit of farming. Anyway, you are not too pretentious – any legendary will do. The possible items are:

- **Shadowmourne** – requires **250 Shards**;
- **Valanyr** – requires **250 Fragments**;
- **Dragonwrath** – requires **250 Motes**;

Shards, Fragments and **Motes** are the **key materials**, all else is **junk.** You will be given lines of input, such as **2 motes 3 ores 15 stones.** Keep track of the **key materials -** the **first** that reaches the **250 mark wins** the **race**. At that point, print the corresponding legendary obtained. Then, print the **remaining** shards, fragments, motes, ordered by **quantity** in **descending** order, each on a new line. Finally, print the collected **junk** items, in **alphabetical** order.

Input

- Each line of input is in format **{quantity} {material} {quantity} {material} … {quantity} {material}**

Output

- On the first line, print the obtained item in format **{Legendary item} obtained!**
- On the next three lines, print the remaining key materials in descending order by quantity
  - If two key materials have the same quantity, print them in alphabetical order
- On the final several lines, print the junk items in alphabetical order
  - All materials are printed in format **{material}: {quantity}**
  - All output should be **lowercase**, except the first letter of the legendary

Constraints

- The quantity-material pairs are between 1 and 25 per line.
- The number of lines is in range [1..10]
- All materials are case-insensitive.
- Allowed working time: 0.25s

- Allowed memory: 16 MB

Examples

Input	Output
3 Motes 5 stones 5 Shards 6 leathers 255 fragments 7 Shards	Valanyr obtained! fragments: 5 shards: 5 motes: 3 leathers: 6 stones: 5

Input	Output
123 silver 6 shards 8 shards 5 motes 9 fangs 75 motes 103 MOTES 8 Shards 86 Motes 7 stones 19 silver	Dragonwrath obtained! shards: 22 motes: 19 fragments: 0 fangs: 9 silver: 123

Problem 13. ** Сръбско Unleashed

Admit it – the СРЪБСКО is your favorite sort of music. You never miss a concert and you have become quite the geek concerning everything involved with СРЪБСКО. You can't decide between all the singers you know who your favorite one is. One way to find out is to keep a statistics of how much money their concerts make. Write a program that does all the boring calculations for you.

On each input line you'll be given data in format: **"singer @venue ticketsPrice ticketsCount"**. There will be **no redundant whitespaces anywhere** in the input. Aggregate the data **by venue and by singer**. For each venue, print the singer and the total amount of money his/her concerts have made on a separate line. **Venues** should be kept in the **same order** they were entered, the **singers** should be **sorted by how much money** they have made in **descending order**. If two singers have made the same amount of money, keep them **in the order** in which **they were entered.**

Keep in mind that if a line is in incorrect format, it should be skipped and its data should not be added to the output. Each of the four tokens must be separated by **a space**, everything else is invalid. The venue should be denoted with **@** in front of it, such as @Sunny Beach

SKIP THOSE: Ceca@Belgrade125 12378, Ceca @Belgrade12312 123

The singer and town name may consist of one to three words.

Input

- The input data should be read from the console.
- It consists of a variable number of lines and ends when the command "**End**" is received.
- The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

- The output should be printed on the console.
- Print the aggregated data for each venue and singer in the format shown below.
- Format for singer lines is **#{2*space}{singer}{space}->{space}{total money}**

Constraints

- The **number of input lines** will be in the range [2 … 50].
- The **ticket price** will be an integer in the range [0 … 200].
- The **ticket count** will be an integer in the range [0 … 100 000]
- **Singers** and **venues** are case sensitive
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
Lepa Brena @Sunny Beach 25 3500 Dragana @Sunny Beach 23 3500 Ceca @Sunny Beach 28 3500 Mile Kitic @Sunny Beach 21 3500 Ceca @Sunny Beach 35 3500 Ceca @Sunny Beach 70 15000 Saban Saolic @Sunny Beach 120 35000 End	Sunny Beach # Saban Saolic -> 4200000 # Ceca -> 1270500 # Lepa Brena -> 87500 # Dragana -> 80500 # Mile Kitic -> 73500

Input	Output
Lepa Brena @Sunny Beach 25 3500 Dragana@Belgrade23 3500 Ceca @Sunny Beach 28 3500 Mile Kitic @Sunny Beach 21 3500 Ceca @Belgrade 35 3500 Ceca @Sunny Beach 70 15000 Saban Saolic @Sunny Beach 120 35000 End	Sunny Beach # Saban Saolic -> 4200000 # Ceca -> 1148000 # Lepa Brena -> 87500 # Mile Kitic -> 73500 Belgrade # Ceca -> 122500

Problem 14. *** Dragon Army

Heroes III is the best game ever. Everyone loves it and everyone plays it all the time. Stamat is no exclusion to this rule. His favorite units in the game are all **types** of dragons – black, red, gold, azure etc… He likes them so much that he gives them **names** and keeps logs of their **stats**: **damage, health** and **armor**. The process of aggregating all the data is quite tedious, so he would like to have a program doing it. Since he is no programmer, it's your task to help him

You need to categorize dragons by their **type**. For each dragon, identified by **name,** keep information about his **stats.** Type is **preserved** as in the order of input, but dragons are **sorted** alphabetically by name. For each type, you should also print the average **damage**, **health** and **armor** of the dragons. For each dragon, print his own stats.

There **may** be **missing** stats in the input, though. If a stat is missing you should assign it default values. Default values are as follows: health **250**, damage **45**, and armor **10**. Missing stat will be marked by **null**.

The input is in the following format **{type} {name} {damage} {health} {armor}.** Any of the integers may be assigned null value. See the examples below for better understanding of your task.

If the same dragon is added a second time, the new stats should **overwrite** the previous ones. Two dragons are considered **equal** if they match by **both** name and type.

Input

- On the first line, you are given number N -> the number of dragons to follow
- On the next N lines you are given input in the above described format. There will be single space separating each element.

Output

- Print the aggregated data on the console
- For each type, print average stats of its dragons in format **{Type}::({damage}/{health}/{armor})**
- Damage, health and armor should be rounded to two digits after the decimal separator
- For each dragon, print its stats in format **-{Name} -> damage: {damage}, health: {health}, armor: {armor}**

Constraints

- N is in range [1…100]
- The dragon type and name are one word only, starting with capital letter.
- Damage health and armor are integers in range [0 … 100000] or **null**

Examples

Input	Output
5 Red Bazgargal 100 2500 25 Black Dargonax 200 3500 18 Red Obsidion 220 2200 35 Blue Kerizsa 60 2100 20 Blue Algordox 65 1800 50	Red::(160.00/2350.00/30.00) -Bazgargal -> damage: 100, health: 2500, armor: 25 -Obsidion -> damage: 220, health: 2200, armor: 35 Black::(200.00/3500.00/18.00) -Dargonax -> damage: 200, health: 3500, armor: 18 Blue::(62.50/1950.00/35.00) -Algordox -> damage: 65, health: 1800, armor: 50 -Kerizsa -> damage: 60, health: 2100, armor: 20

Input	Output
4 Gold Zzazx null 1000 10 Gold Traxx 500 null 0 Gold Xaarxx 250 1000 null Gold Ardrax 100 1055 50	Gold::(223.75/826.25/17.50) -Ardrax -> damage: 100, health: 1055, armor: 50 -Traxx -> damage: 500, health: 250, armor: 0 -Xaarxx -> damage: 250, health: 1000, armor: 10 -Zzazx -> damage: 45, health: 1000, armor: 10

Problems with matrices

Problem 15. Rubik's Matrix

Rubik's cube – everyone's favorite head-scratcher. Writing a program to solve it will be quite a difficult task for an exam, though. Instead, we have a Rubik's matrix prepared for you. You will be given a pair of dimensions: **R** and **C.** To prepare the matrix, fill each row with increasing integers, starting from one. For example, **2 x 4** matrix must look like this:

1	2	3	4
5	6	7	8

Next, you will receive series of commands, indicating which row or column you must move, in which direction, and how many times. For example, **1 up 1** means: column 1, direction: up, 1 move. After executing it, the matrix should look like:

1	6	3	4
5	2	7	8

Directions **left** and **right** mean you must move the corresponding **row**, while **up** and **down** and related to the **columns.** After shuffling the Rubik's matrix, you have to **rearrange** it (meaning that the **values in each cell** should be in **increasing order**, such as the ones in the original matrix). The rearranging process should start at **top-left** and end at **bottom-right**. Find the **position** of the value you need, and print the **swap command** on the console, in the format described below.

Input

- On the first line, you are given the integers **R** and **C**, separated by a space.
- On the second line, you are given an integer **N**, indicating the number of commands to follow
- On the next N lines, you are given commands in format **{row/col} {direction} {moves}**

Output

- On **R** * **C** number of lines, print the **swap commands** needed to rearrange the matrix, either:
  - **Swap ({row}, {col}) with ({row}, {col})** or
  - **No swap required**

Constraints

- **R, C, N** are integers in range [1 … 100]
- {**row**} and {**col**} will always be inside the matrix
- {**moves**} is in range [0 … 2^{31}-1]
- Allowed time and memory: 0.25s / 16 MB

Examples

Input	Output	Input	Output
3 3 2 1 down 1 1 left 1	No swap required Swap (0, 1) with (1, 0) No swap required Swap (1, 0) with (1, 2) Swap (1, 1) with (2, 1) Swap (1, 2) with (2, 1) No swap required No swap required No swap required	3 3 2 0 down 3 0 left 3	No swap required No swap required No swap required No swap required No swap required No swap required No swap required No swap required No swap required

Problem 16. Target Practice

Cotton-eye Gosho has a problem. Snakes! An infestation of snakes! Gosho is a red-neck which means he doesn't really care about animal rights, so he bought some ammo, loaded his shotgun and started shooting at the poor creatures. He has a favorite spot – rectangular stairs which the snakes like to climb in order to reach Gosho's stash of whiskey in the attic. You're tasked with the horrible cleanup of the aftermath.

A **snake** is represented by **a string**. The **stairs** are a **rectangular matrix of size NxM**. A snake starts climbing the stairs from the **bottom-right corner** and slithers its way up in a **zigzag** – first it moves left until it reaches the left wall, it climbs on the next row and starts moving right, then on the third row, moving left again and so on. The first cell (bottom-right corner) is filled with the first symbol of the snake, the second cell (to the left of the first) is filled with the second symbol, etc. The snake is as long as it takes in order to **fill the stairs completely** – if you reach the end of the string representing the snake, start again at the beginning. Gosho is patient and a sadist, he'll wait until the stairs are completely covered with snake and will then fire a shot.

The shot has three parameters – **impact row, impact column and radius**. When the projectile lands on the specified coordinates of the matrix it **destroys all symbols in the given radius (turns them into a space)**. You can check whether a cell is inside the blast radius using the Pythagorean Theorem (very similar to the "point inside a circle" problem).

The symbols above the impact area start **falling down until they land on other symbols (meaning a symbol moves down a row as long as there is a space below)**. When the horror ends, print on the console the **resulting staircase, each row on a new line**. You should really check out the examples at this point.

Input

- The input data should be read from the console. It consists of exactly three lines.
- On the first line, you'll receive the **dimensions** of the stairs in format: **"N M"**, where **N** is the number of **rows**, and **M** is the number of **columns**. They'll be separated by a single space.
- On the second line you'll receive the string representing the **snake**.
- On the third line, you'll receive the **shot parameters (impact row, impact column and radius)**, all separated by a **single space**.
- The input data will always be valid and in the format described. There is no need to check it explicitly.

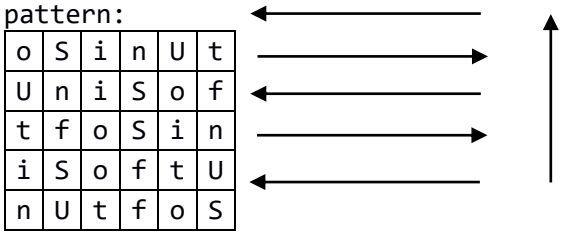
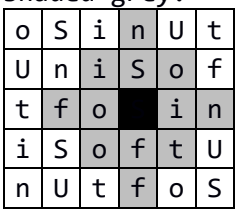
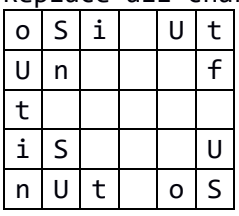
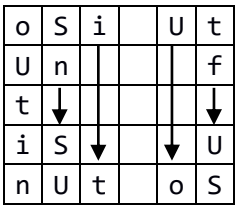
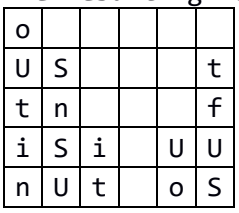
Output

- The output should be printed on the console. It should consist of **N lines**.
- Each line should contain a string representing the respective row of the final matrix.

Constraints

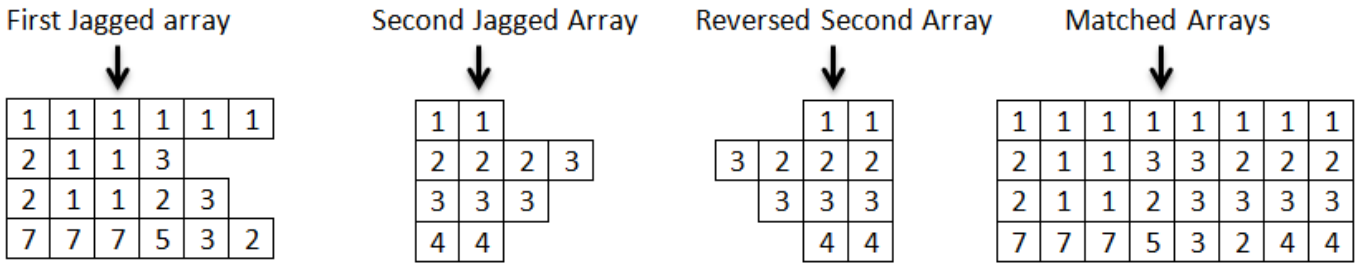
- The **dimensions** N and M of the matrix will be integers in the range [1 … 12].
- The **snake** will be a string with length in the range [1 … 20] and **will not contain any whitespace characters**.
- The shot's **impact row and column** will always be **valid coordinates** in the matrix – they will be integers in the range [0 … N – 1] and [0 … M – 1] respectively.
- The shot's **radius** will be an integer in the range [0 … 4].
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output	Comments
5 6 SoftUni 2 3 2	o US t tn f iSi UU nUt oS	The matrix has 5 rows and 6 columns. Fill it in the described pattern:  The shot lands on cell (2,3). It has a radius of 2 cells. The impact cell is shaded black and the other cells within the shot radius are shaded grey.  Replace all characters in the blast area with a space:  All characters start falling down until they land on other characters:  The resulting matrix is: 

Problem 17. Lego Blocks

You are given two **jagged arrays**. Each array represents a **Lego block** containing integers. Your task is first to **reverse** the second jagged array and then check if it would **fit perfectly** in the first jagged array.



The picture above shows exactly what fitting arrays mean. If the arrays fit perfectly you should **print out** the newly made rectangular matrix. If the arrays do not match (they do not form a rectangular matrix) you should print out the **number of cells** in the first array and in the second array combined. The examples below should help you understand more the assignment.

Input

The first line of the input comes as an **integer number n** saying how many rows are there in both arrays. Then you have **2 * n** lines of numbers separated by whitespace(s). The first **n** lines are the rows of the first jagged array; the next **n** lines are the rows of the second jagged array. There might be leading and/or trailing whitespace(s).

Output

You should print out the combined matrix in the format:

```
[elem, elem, …, elem]
[elem, elem, …, elem]
[elem, elem, …, elem]
```

If the two arrays do not fit you should print out : **`The total number of cells is: count`**

Constraints

- The number n will be in the range [2…10].
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output
2 1 1 1 1 1 1 2 1 1 3 1 1 2 2 2 3	[1, 1, 1, 1, 1, 1, 1, 1] [2, 1, 1, 3, 3, 2, 2, 2]
2 1 1 1 1 1 1 1 1 1 1 1 1 1 1	The total number of cells is: 14

Problem 18. Radioactive Mutant Vampire Bunnies

Browsing through GitHub, you come across an old JS Basics teamwork game. It is about very nasty bunnies that multiply extremely fast. There's also a player that has to escape from their lair. You really like the game so you decide to port it to C# because that's your language of choice. The last thing that is left is the algorithm that decides if the player will escape the lair or not.

The **bunnies** are marked with **B,** the **player** is marked with P, and **everything** else is free space, marked with a dot (.) First, you will receive a line holding integers **N** and **M**, which represent the rows and columns in the lair. Then you receive **N** strings that can **only** consist of dots (.), bunnies (B), and the player (P). They represent the initial state of the lair. There will be **only** one player. Then you will receive a string with **commands** such as **LLRRUUDD** – where each letter represents the next **step** of the player (Left, Right, Up, Down).

After each step of the player, the bunnies spread to the up, down, left and right (neighboring cells marked as "." **change** their value to B). If the player **moves** to a bunny cell or a bunny **reaches** the player, the player has died. If the player goes **out** of the lair **without** encountering a bunny, the player has won.

If the player **dies** or **wins**, the game ends. All the activities for **this** turn continue (e.g. all the bunnies spread normally), but there are no more turns. There will be **no** stalemates where the moves of the player end before he dies or escapes.

Print the final state of the lair with every row on a separate line. On the last line, print either **"dead: {row} {col}"** or **"won: {row} {col}"**. Row and col are the coordinates of the cell where the player has died or the last cell he has been in before escaping the lair.

Input

- On the first line of input, the number N and M are received – the number of rows and columns in the lair
- On the next N lines, each row is received in the form of a string. The string will contain only ".", "B", "P". All strings will be the same length. There will be only one "P" for all the input
- On the last line, the directions are received in the form of a string, containing "R", "L", "U", "D"

Output

- On the first N lines, print the final state of the bunny lair
- On the last line, print the outcome – "won:" or "dead:" + {row} {col}

Constraints

- The dimensions of the lair are in range [3…20]
- The directions string length is in range [1..20]

Examples

Input	Output
5 8 ....B ...B.... ....B..B ....P..... ULLL	BBBBBBBB BBBBBBBB BBBBBBBB .BBBBBB ..BBBBB won: 3 0

Input	Output
4 5 .... .B... ...P. LLLLLLLL	.B... BBB.. BBBB. BBB.. dead: 3 1

Problem 19.* Crossfire

You will receive two integers which represent the dimensions of a matrix. Then, you must fill the matrix with increasing integers starting from 1, and continuing on every row, like this:
first row: 1, 2, 3… n
second row: n+1, n+2, n+3… n + n.

You will also receive several commands in the form of 3 integers separated by a space. Those 3 integers will represent a row in the matrix, a column and a radius. You must then **destroy** the cells which correspond to those arguments **cross-like.**

Destroying a cell means that, **that** cell becomes completely **nonexistent** in the matrix. Destroying cells **cross-like** means that you form a **cross figure**, with center point - equal to the cell with coordinates – the **given row** and **column**, and **lines** with length equal to the **given radius**. See the examples for more info.

The input ends when you receive the command “Nuke it from orbit”. When that happens, you must print what has remained from the initial matrix.

Input

- On the first line you will receive the dimensions of the matrix. You must then fill the matrix according to those dimensions.
- On the next several lines you will begin receiving 3 integers separated by a single **space**, which represent the row, col and radius. You must then destroy cells according to those coordinates.
- When you receive the command “Nuke it from orbit” the input ends.

Output

- The output is simple. You must print what is left from the matrix.
- Every row must be printed on a new line and every column of a row - separated by a space.

Constraints

- The dimensions of the matrix will be integers in the range [2, 100].
- The given rows and columns will be valid integers in the range [$-2^{31} + 1$, $2^{31} - 1$].
- The radius will be in range [0, $2^{31} - 1$].
- Allowed time/memory: 250ms/16MB.

Examples

Input	Output	Comment
5 5 3 3 2 4 3 2 Nuke it from orbit	1 2 3 4 5 6 7 8 10 11 12 13 16 21	Initial matrix: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 Result from first destruction: 1 2 3 4 5 6 7 8 10 11 12 13 15 16 21 22 23 25 Result from second destruction: 1 2 3 4 5 6 7 8 10 11 12 13 16 21

Input	Output
5 5 4 4 4 Nuke it from orbit	1 2 3 4 6 7 8 9 11 12 13 14 16 17 18 19

Problem 20.* The Heigan Dance

At last, level 80. And what do level eighties do? Go raiding. This is where you are now – trying not to be wiped by the famous dance boss, Heigan the Unclean. The fight is pretty straightforward - dance around the Plague Clouds and Eruptions, and you'll be just fine.

Heigan's chamber is a 15-by-15 two-dimensional array. The player always starts at the **exact center.** For each turn, Heigan uses a spell that hits a certain cell and the neighboring **rows/columns**. For example, if he hits (1,1), he also hits (0,0, 0,1, 0,2, 1,0 … 2,2). If the player's current position is within the area of damage, the player tries to move. First he tries to move **up**, if there's **damage/wall**, he tries to move **right**, then **down**, then **left**. If he **cannot move** in any direction, because **the cell is damaged** or there is **a wall**, the player **stays** in place and takes the damage.

Plague cloud does 3500 damage **when it hits**, and 3500 damage **the next turn**. Then it **expires. Eruption** does 6000 damage **when it hits.** If a spell will hit a player that also has an active Plague Cloud from the previous turn, the **cloud** damage is applied **first**. **Both** Heigan and the player **may** die in the same turn. If Heigan is **dead**, the spell he **would** have casted is **ignored**.

The player always starts at **18500** hit points; Heigan starts at **3,000,000** hit points. **Each** turn, the player does damage to Heigan. The fight is over either when the player is **killed**, or Heigan is **defeated**.

Input

- On the first line you receive a floating-point number **D –** the damage done to Heigan each turn

- On the next several lines – you receive input in format **{spell} {row} {col}** – **{spell}** is either **Plague** or **Eruption**

Output

- On the first line
  - If Heigan is defeated: "**Heigan: Defeated!**"
  - Else: "**Heigan: {remaining}**", where remaining is rounded to two digits after the decimal separator
- On the second line:
  - If the player is killed: "**Player: Killed by {spell}**"
  - Else "**Player: {remaining}"**
- On the third line: "**Final position: {row, col}**" -> the last coordinates of the player.

Constraints

- **D** is a floating-point number in range [0 … 500000]
- A damaging spell will always affect at least one cell
- Allowed memory: 16 MB
- Allowed working time: 0.25s

Examples

Input	Output
10000 Cloud 7 7 Eruption 6 7 Eruption 8 7 Eruption 8 7	Heigan: 2960000.00 Player: Killed by Eruption Final position: 8, 7

Input	Output
500000 Cloud 7 6 Eruption 7 8 Eruption 7 7 Cloud 7 8 Eruption 7 9 Eruption 6 14 Eruption 7 11	Heigan: Defeated! Player: 12500 Final position: 7, 11

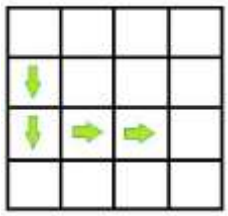
Input	Output
12500.66 Cloud 7 7 Cloud 7 7 Cloud 7 7 Cloud 7 7	Heigan: 2949997.36 Player: Killed by Plague Cloud Final position: 7, 7

Problem 21.** Parking System

The parking lot in front of SoftUni is one of the busiest in the country, and it's a common cause for conflicts between the doorkeeper Bai Tzetzo and the students. The SoftUni team wants to proactively resolve all conflicts, so an automated parking system should be implemented. They are organizing a competition – Parkoniada – and the author of the best parking system will win a romantic dinner with RoYaL. That's **exactly** what you've been dreaming of, so you decide to join in.

The parking lot is a **rectangular** matrix where the **first** column is **always** free and all other cells are parking spots. A car can enter from any cell of the first column and then decides to go to a specific spot. If that spot is **not** free, the car searches for the **closest** free spot on the **same** row. If **all** the cells on that specific row are used, the car cannot park and leaves. If **two** free cells are located at the **same** distance from the **initial** parking spot, the cell which is **closer** to the entrance is preferred. A car can **pass** through a used parking spot.

Your task is to calculate the distance travelled by each car to its parking spot.



A car enters the parking at row 1. It wants to go to cell 2, 2 so it moves through **exactly four** cells to reach its parking spot.

Input

- On the first line of input, you are given the integers **R** and **C**, defining the dimensions of the parking lot.
- On the next several lines, you are given the integers **Z, X, Y** where **Z** is the entry row and **X, Y** are the coordinates of the desired parking spot.
- The input stops with the command '**stop**'. All integers are separated by a **single** space.

Output

- For each car, print the distance travelled to the desired spot or the first free spot.
- If a car cannot park on its desired row, print the message '**Row {row number} full**'

Constraints

- 2 ≤ R,C ≤ 10000
- Z, X, Y are inside the dimensions of the matrix. Y is never on the first column.
- There are no more than 1000 input lines
- Allowed time/space: 0.1s (C#) 0.25s (Java)/16MB

Input	Output
4 4 1 2 2 2 2 2 2 2 2 3 2 2 stop	4 2 4 Row 2 full