# Programming Fundamentals Final Exam Retake 15.08.2020

## Problem 2. Ad Astra

*You are an astronaut who just embarked on a mission across the solar system. Since you will be in space for a long time, you have packed a lot of food with you. Create a program, which helps you identify how much food you have left and gives you information about its expiration date.*

On the first line of the input you will be given a **text string**. You must extract the information about the food **and calculate the total calories.**

First you must **extract the food info**. It will always follow the same pattern rules:

* It will be surrounded by "|" or "#" (only one of the two) in the following pattern:   
  #{item name}#{expiration date}#{calories}# or   
  |{item name}|{expiration date}|{calories}|
* The item name will contain **only lowercase and uppercase letters and whitespace**
* The expiration date will always follow the pattern: {day}/{month}/{year}**, where the day, month and year will be exactly two digits long**
* The calories will be **an integer between 0-10000**

Calculate **the total calories of all food items** and then determine **how many days you can last with the food you have**. Keep in mind that **you need 2000kcal a day**.

### Input / Constraints

* You will receive **a single string**

### Output

* First print **the amount of days** you will be able to last with the food you have:

**"You have food to last you for: {days} days!"**

* **The output for each food item should look like this:  
  "Item: {item name}, Best before: {expiration date}, Nutrition: {calories}"**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | | |
| #Bread#19/03/21#4000#|Invalid|03/03.20||Apples|08/10/20|200||Carrots|06/08/20|500||Not right|6.8.20|5| | | |
| **Output** | | **Comments** |
| You have food to last you for: 2 days!  Item: Bread, Best before: 19/03/21, Nutrition: 4000  Item: Apples, Best before: 08/10/20, Nutrition: 200  Item: Carrots, Best before: 06/08/20, Nutrition: 500 | | We have a total of three matches – bread, apples and carrots.  The sum of their calories is 4700. Since you need 2000kcal a day, we divide 4700/2000, which means this food will last you for 2 days.  We print each item |
| **Input** | | |
| $$#@@%^&#Fish#24/12/20#8500#|#Incorrect#19.03.20#450|$5\*(@!#Ice Cream#03/10/21#9000#^#@aswe|Milk|05/09/20|2000| | | |
| **Output** | | **Comments** |
| You have food to last you for: 9 days!  Item: Fish, Best before: 24/12/20, Nutrition: 8500  Item: Ice Cream, Best before: 03/10/21, Nutrition: 9000  Item: Milk, Best before: 05/09/20, Nutrition: 2000 | | We have three matches. The total calories are 8500 + 9000 + 2000 = 19500, which means you have food for a total of 9 days. |
| **Input** | | |
| Hello|#Invalid food#19/03/20#450|$5\*(@ | | |
| **Output** | **Comments** | |
| You have food to last you for: 0 days! | We have no matches, which means we have no food. The colored text is not a match, since it doesn't have a # at the end. | |

|  |  |
| --- | --- |
| **JavaScript Input** | |
| [  '#Bread#19/03/21#4000#|Invalid|03/03.20||Apples|08/10/20|200||Carrots|06/08/20|500||Not right|6.8.20|5|'  ] | |
| **Output** | **Comments** |
| You have food to last you for: 2 days!  Item: Bread, Best before: 19/03/21, Nutrition: 4000  Item: Apples, Best before: 08/10/20, Nutrition: 200  Item: Carrots, Best before: 06/08/20, Nutrition: 500 | We have a total of three matches – bread, apples and carrots.  The sum of their calories is 4700. Since you need 2000kcal a day, we divide 4700/2000, which means this food will last you for 2 days.  We print each item |
| **JavaScript Input** | |
| [ '$$#@@%^&#Fish#24/12/20#8500#|#Incorrect#19.03.20#450|$5\*(@!#Ice Cream#03/10/21#9000#^#@aswe|Milk|05/09/20|2000|' ] | |
| **Output** | **Comments** |
| You have food to last you for: 9 days!  Item: Fish, Best before: 24/12/20, Nutrition: 8500  Item: Ice Cream, Best before: 03/10/21, Nutrition: 9000  Item: Milk, Best before: 05/09/20, Nutrition: 2000 | We have three matches. The total calories are 8500 + 9000 + 2000 = 19500, which means you have food for a total of 9 days. |
| **JavaScript Input** | |
| ['Hello|#Invalid food#19/03/20#450|$5\*(@' ] | |
| **Output** | **Comments** |
| You have food to last you for: 0 days! | We have no matches, which means we have no food. The colored text is not a match, since it doesn't have a # at the end. |

# Programming Fundamentals Final Exam Retake 15.08.2020

## Problem 3. Pianist

*You are a pianist and you like to keep a list of your favorite piano pieces. Create a program, to help you organize it and add, change, remove pieces from it!*

On the first line of the standard input you will receive an integer **n** – the **number of pieces** that you will initially have. On the next **n** lines the **pieces themselves** will follow with their **composer** and **key**, separated by "|" in the following format:

{piece}|{composer}|{key}

Then, you will be receiving different **commands**, each on a new line, separated by "|", until the "Stop" command is given:

* Add|{piece}|{composer}|{key}
  + You need to **add the given piece** with the information about it to the other pieces
  + If the piece **is already in the collection**, print:

"**{piece} is already in the collection!**"

* + If the piece is **not in the collection**, print:   
    "{piece} by {composer} in {key} added to the collection!"
* Remove|{piece}
  + If the piece is in the collection, **remove it** and print:

"Successfully removed {piece}!"

* + If the piece is **not in the collection**, print:

"Invalid operation! {piece} does not exist in the collection."

* ChangeKey|{piece}|{new key}
  + If the piece is in the collection, **change its key with the given one** and print:

"Changed the key of {piece} to {new key}!"

* + If the piece is **not in the collection**, print:

"Invalid operation! {piece} does not exist in the collection."

Upon receiving the "Stop" command you need to print all pieces in your collection, sorted by their **name and by the name of their composer in alphabetical order**, in the following format:  
"**{Piece} -> Composer: {composer}, Key: {key}**"

### Input/Constraints

* You will receive **a single integer** at first – **the initial number of pieces in the collection**
* For each piece you will receive a single line of text with information about it.
* Then you will receive multiple commands in the way described above, until the command "Stop".

### Output

* All the output messages with the appropriate formats are described in the problem description.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Fur Elise|Beethoven|A Minor  Moonlight Sonata|Beethoven|C# Minor  Clair de Lune|Debussy|C# Minor  Add|Sonata No.2|Chopin|B Minor  Add|Hungarian Rhapsody No.2|Liszt|C# Minor  Add|Fur Elise|Beethoven|C# Minor  Remove|Clair de Lune  ChangeKey|Moonlight Sonata|C# Major  Stop | Sonata No.2 by Chopin in B Minor added to the collection!  Hungarian Rhapsody No.2 by Liszt in C# Minor added to the collection!  Fur Elise is already in the collection!  Successfully removed Clair de Lune!  Changed the key of Moonlight Sonata to C# Major!  Fur Elise -> Composer: Beethoven, Key: A Minor  Hungarian Rhapsody No.2 -> Composer: Liszt, Key: C# Minor  Moonlight Sonata -> Composer: Beethoven, Key: C# Major  Sonata No.2 -> Composer: Chopin, Key: B Minor |
| **Comments** | |
| After we receive the initial pieces with their info, we start receiving commands. The first two commands are to add a piece to the collection and since the pieces are not already added, we manage to add them. The third add command, however, **attempts to add a piece, which is already in the collection**, so we print a special message and don't add the piece. After that we receive the remove command and since the piece is in the collection, we remove it successfully. Finally, the last command says to change the key of a piece. Since the key is present in the collection, we modify its key.  We receive the Stop command, print the information about the pieces, sorted in the way described above, and the program ends. | |
| **Input** | **Output** |
| 4  Eine kleine Nachtmusik|Mozart|G Major  La Campanella|Liszt|G# Minor  The Marriage of Figaro|Mozart|G Major  Hungarian Dance No.5|Brahms|G Minor  Add|Spring|Vivaldi|E Major  Remove|The Marriage of Figaro  Remove|Turkish March  ChangeKey|Spring|C Major  Add|Nocturne|Chopin|C# Minor  Stop | Spring by Vivaldi in E Major added to the collection!  Successfully removed The Marriage of Figaro!  Invalid operation! Turkish March does not exist in the collection.  Changed the key of Spring to C Major!  Nocturne by Chopin in C# Minor added to the collection!  Eine kleine Nachtmusik -> Composer: Mozart, Key: G Major  Hungarian Dance No.5 -> Composer: Brahms, Key: G Minor  La Campanella -> Composer: Liszt, Key: G# Minor  Nocturne -> Composer: Chopin, Key: C# Minor  Spring -> Composer: Vivaldi, Key: C Major |

*Suggestion: Choose a piece from the ones here to listen while you are doing the problem!*

|  |  |
| --- | --- |
| **JavaScript Input** | **Output** |
| [  '3',  'Fur Elise|Beethoven|A Minor',  'Moonlight Sonata|Beethoven|C# Minor',  'Clair de Lune|Debussy|C# Minor',  'Add|Sonata No.2|Chopin|B Minor',  'Add|Hungarian Rhapsody No.2|Liszt|C# Minor',  'Add|Fur Elise|Beethoven|C# Minor',  'Remove|Clair de Lune',  'ChangeKey|Moonlight Sonata|C# Major',  'Stop'  ] | Sonata No.2 by Chopin in B Minor added to the collection!  Hungarian Rhapsody No.2 by Liszt in C# Minor added to the collection!  Fur Elise is already in the collection!  Successfully removed Clair de Lune!  Changed the key of Moonlight Sonata to C# Major!  Fur Elise -> Composer: Beethoven, Key: A Minor  Hungarian Rhapsody No.2 -> Composer: Liszt, Key: C# Minor  Moonlight Sonata -> Composer: Beethoven, Key: C# Major  Sonata No.2 -> Composer: Chopin, Key: B Minor |
| **Comments** | |
| After we receive the initial pieces with their info, we start receiving commands. The first two commands are to add a piece to the collection and since the pieces are not already added, we manage to add them. The third add command, however, **attempts to add a piece, which is already in the collection**, so we print a special message and don't add the piece. After that we receive the remove command and since the piece is in the collection, we remove it successfully.  Finally, the last command says to change the key of a piece. Since the key is present in the collection, we modify its key.   We receive the Stop command, print the information about the pieces, sorted in the way described above, and the program ends. | |
| **JavaScript Input** | **Output** |
| [  '4',  'Eine kleine Nachtmusik|Mozart|G Major',  'La Campanella|Liszt|G# Minor',  'The Marriage of Figaro|Mozart|G Major',  'Hungarian Dance No.5|Brahms|G Minor',  'Add|Spring|Vivaldi|E Major',  'Remove|The Marriage of Figaro',  'Remove|Turkish March',  'ChangeKey|Spring|C Major',  'Add|Nocturne|Chopin|C# Minor',  'Stop'  ] | Spring by Vivaldi in E Major added to the collection!  Successfully removed The Marriage of Figaro!  Invalid operation! Turkish March does not exist in the collection.  Changed the key of Spring to C Major!  Nocturne by Chopin in C# Minor added to the collection!  Eine kleine Nachtmusik -> Composer: Mozart, Key: G Major  Hungarian Dance No.5 -> Composer: Brahms, Key: G Minor  La Campanella -> Composer: Liszt, Key: G# Minor  Nocturne -> Composer: Chopin, Key: C# Minor  Spring -> Composer: Vivaldi, Key: C Major |

# Programming Fundamentals Final Exam Retake 15.08.2020

## Problem 1. The Imitation Game

*You are a mathematician during world war 2, who has joined the cryptography team to decipher the enemy's enigma code. Your job is to create a program to crack the codes.*

On the first line of the input you will receive the **encrypted message**. After that, until the "Decode" command is given, **you will be receiving strings** with **instructions** for different **operations** that need to be performed upon the **concealed message** to **interpret** **it** and reveal its true content. There are several types of instructions, split by '|'

* Move {number of letters}
  + Moves the first n letters to the back of the string.
* Insert {index} {value}
  + Inserts the given value before the given index in the string.
* ChangeAll {substring} {replacement}
  + Changes all occurrences of the given substring with the replacement text.

### Input / Constraints

* On the first line, you will receive a string with message.
* On the next lines, you will be receiving commands, split by **'|' .**

### Output

* After the "Decode" command is received, print this message:  
  "**The decrypted message is: {message}**"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| zzHe  ChangeAll|z|l  Insert|2|o  Move|3  Decode | The decrypted message is: Hello |
| **Comments** | |
| **ChangeAll|z|l**  zzHe → llHe (We replace all occurrences of 'z' with 'l')  **Insert|2|o**  llHe → lloHe (We add an 'o' before the character on index 2)  Move|3  lloHe → Hello (We take the first three characters and move them to the end of the string)  Finally, after receiving the **"Decode"** command, we print the resulting message. | |
| **Input** | **Output** |
| owyouh  Move|2  Move|3  Insert|3|are  Insert|9|?  Decode | The decrypted message is: howareyou? |

|  |  |
| --- | --- |
| **JavaScript Input** | **Output** |
| [  'zzHe',  'ChangeAll|z|l',  'Insert|2|o',  'Move|3',  'Decode'  ] | The decrypted message is: Hello |
| **Comments** | |
| **ChangeAll|z|l**  zzHe → llHe (We replace all occurrences of 'z' with 'l')  **Insert|2|o**  llHe → lloHe (We add an 'o' before the character on index 2)  Move|3  lloHe → Hello (We take the first three characters and move them to the end of the string)  Finally, after receiving the **"Decode"** command, we print the resulting message. | |
| **JavaScript Input** | **Output** |
| [  'owyouh',  'Move|2',  'Move|3',  'Insert|3|are',  'Insert|9|?'  'Decode'  ] | The decrypted message is: howareyou? |

# Programming Fundamentals Final Exam 09.08.2020

## Problem 2. Destination Mapper

*Now that you have planned out your tour, you are ready to go! Your next task is to mark all the points on the map that you are going to visit.*

You will be given a **string** representing some **places** on the map. You have to **filter** only the **valid ones**. A valid location is:

* Surrounded by **"="** or **"/"** on **both sides** (the **first** and the **last** symbols must **match**)
* After the **first "="** or **"/"** there should be **only letters** (the **first** must be **upper-case**)
* The **letters** must be **at least 3**

**Example**: In the string **"=Hawai=/Cyprus/=Invalid/invalid==i5valid=/I5valid/=i="** only the **first two** locations are valid.

After you have **matched** all the **valid locations**, you have to **calculate travel points**. They are calculated by **summing** the **lengths** of all the **valid destinations** that you have found on the map. At the end, on the **first line** print the following: **"Destinations: {destinations joined by ', '}"**. On the **second line** print **"Travel Points: {travel\_points}"**.

### Input / Constraints

* You will recive a string representing the **locations on the map**
* **JavaScript**: you will receive a **single parameter: string**

### Output

* Print the **messages described above**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| =Hawai=/Cyprus/=Invalid/invalid==i5valid=/I5valid/=i= | Destinations: Hawai, Cyprus  Travel Points: 11 |
| ThisIs some InvalidInput | Destinations:  Travel Points: 0 |

# Programming Fundamentals Final Exam 09.08.2020

## Problem 3. Plant Discovery

*You have now returned from your world tour. On your way you have discovered some new plants and you want to gather some information about them and create an exhibition to see which plant is highest rated.*

On the **first line** you will receive a number **n**. On the next **n lines**, you will be given some information about the plants that you have discovered in the format: **"{plant}<->{rarity}"**. **Store** that **information**, because you will need it later. If you receive a plant **more than once**, **update** its rarity.

After that until you receive the **command** **"Exhibition"** you will be given some of these **commands**:

* **Rate: {plant} - {rating}** – **add** the given **rating** to the plant (**store all ratings**)
* **Update: {plant} - {new\_rarity}** – **update** the **rarity** of the plant with the **new one**
* **Reset: {plant}** – **remove all** the **ratings** of the given plant

***Note: If any of the command is invalid, print "error"***

After the command **"Exhibition"** print the information that you have about the plants in the following format:

**Plants for the exhibition:  
- {plant\_name}; Rarity: {rarity}; Rating: {average\_rating formatted to the 2nd digit}  
…**

The plants should be **sorted** by **rarity descending**, then by **average rating descending**

### Input / Constraints

* You will recive the input as **described above**
* **JavaScript**: you will receive a **list of strings**

### Output

* Print the **information** about all plants as **described above**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Arnoldii<->4  Woodii<->7  Welwitschia<->2  Rate: Woodii - 10  Rate: Welwitschia - 7  Rate: Arnoldii - 3  Rate: Woodii - 5  Update: Woodii - 5  Reset: Arnoldii  Exhibition | Plants for the exhibition:  - Woodii; Rarity: 5; Rating: 7.50  - Arnoldii; Rarity: 4; Rating: 0.00  - Welwitschia; Rarity: 2; Rating: 7.00 |
| 2  Candelabra<->10  Oahu<->10  Rate: Oahu - 7  Rate: Candelabra - 6  Exhibition | Plants for the exhibition:  - Oahu; Rarity: 10; Rating: 7.00  - Candelabra; Rarity: 10; Rating: 6.00 |

# Programming Fundamentals Final Exam 09.08.2020

## Problem 1. World Tour

*You are a world traveller and your next goal is to make a world tour. In order to do that, you have to plan out everything first. To start with, you would like to plan out all of your stops where you will have a break.*

On the **first line** you will be given a string containing all of your **stops**. Until you receive the command **"Travel"**, you will be given some commands to **manipulate** that initial string. The **commands can be**:

* **Add Stop:{index}:{string}** – **insert** the given **string** at that **index** only if the index **is valid**
* **Remove Stop:{start\_index}:{end\_index}** – **remove** the elements of the string from the **starting index** to the **end index** (**inclusive**) if **both** indices are **valid**
* **Switch:{old\_string}:{new\_string}** – if the **old string** is in the initial string, **replace** it with the **new one**. (all **occurrences**)

***Note: After each command print the current state of the string***

After the **"Travel"** command, print the following: **"Ready for world tour! Planned stops: {string}"**

### Input / Constraints

* **JavaScript**: you will receive a **list of strings**

### Output

* Print the proper output messages in the proper cases as described in the problem description

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Hawai::Cyprys-Greece  Add Stop:7:Rome  Remove Stop:11:16  Switch:Hawai:Bulgaria  Travel | Hawai::RomeCyprys-Greece  Hawai::Rome-Greece  Bulgaria::Rome-Greece  Ready for world tour! Planned stops: Bulgaria::Rome-Greece |

# Programming Fundamentals Final Exam Retake 10.04.2020

## Problem 3. Need for Speed III

*You have just bought the latest and greatest computer game – Need for Seed III. We know that you can`t wait to start playing. Pick your favorite cars and drive them all you want!*

On the first line of the standard input you will receive an integer **n** – the **number of cars** that you can obtain. On the next **n** lines the **cars themselves** will follow with their **mileage** and **fuel** **available**, separated by "|" in the following format:

{car}|{mileage}|{fuel}

Then, you will be receiving different **commands**, each on a new line, separated by " : ", until the "Stop" command is given:

* Drive : {car} : {distance} : {fuel}
  + You need to **drive the given distance** and you will **need the given** fuel to do that. If the car **doesn`t have enough fuel** print:  
    "**Not enough fuel to make that ride**"
  + If the car has the required fuel available in the tank, **increase its mileage** with **the given distance**, **decrease its fuel with the given fuel** and **print**:   
    "{car} driven for {distance} kilometers. {fuel} liters of fuel consumed."
  + You like driving new cars only, so if the mileage of a car reaches **100 000** km, remove it from the collection(s). Print:  
    "**Time to sell the {car}!**"
* Refuel : {car} : {fuel}
  + Refill the tank of your car.
  + Each tank can hold a **maximum of 75 liters of fuel**, so if the given amount of fuel is more than you can fit in the tank, take only what is required to fill it up.
  + Print a message in the following format:  
    "{car} refueled with {fuel} liters"
* Revert : {car} : {kilometers}
  + Decrease the **mileage** of the given **car with the given kilometers** and print the kilometers you have decreased it with in the following format:  
    "{car} mileage decreased by {amount reverted} kilometers"
  + If the mileage becomes **less** **than** **10 000km** **after** it is decreased, **just set it to 10 000km** and   
    **DO NOT print anything.**

Upon receiving the "Stop" command you need to print all cars in your possession, sorted by their **mileage in decscending order**, then by their **name in ascending order**, in the following format:  
"**{car} -> Mileage: {mileage} kms, Fuel in the tank: {fuel} lt.**"

### Input/Constraints

* The **mileage** and **fuel** of the cars will be valid, 32-bit integers and will never be negative.
* The **fuel** and **distance** amounts **in the commands will never be negative**.
* The **car** **names** in the **commands** will always be **valid cars in your possession**.

### Output

* All the output messages with the appropriate formats are described in the problem description.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Audi A6|38000|62  Mercedes CLS|11000|35  Volkswagen Passat CC|45678|5  Drive : Audi A6 : 543 : 47  Drive : Mercedes CLS : 94 : 11  Drive : Volkswagen Passat CC : 69 : 8  Refuel : Audi A6 : 50  Revert : Mercedes CLS : 500  Revert : Audi A6 : 30000  Stop | Audi A6 driven for 543 kilometers. 47 liters of fuel consumed.  Mercedes CLS driven for 94 kilometers. 11 liters of fuel consumed.  Not enough fuel to make that ride  Audi A6 refueled with 50 liters  Mercedes CLS mileage decreased by 500 kilometers  Volkswagen Passat CC -> Mileage: 45678 kms, Fuel in the tank: 5 lt.  Mercedes CLS -> Mileage: 10594 kms, Fuel in the tank: 24 lt.  Audi A6 -> Mileage: 10000 kms, Fuel in the tank: 65 lt. |
| **Comments** | |
| After we receive the cars with their mileage and fuel, we start driving them. When we get to "**Drive : Volkswagen Passat CC : 69 : 8**" command, our program calculates that there is not enough fuel and we print the appropriate message. Then we refuel the Audi A6 with 50 l of fuel and Revert the Mercedes with 500 kilometers.  When we receive the "Revert : Audi A6 : 30000", we set its mileage to **10000** km, because if the current mileage of the Audi is **38543** kms and if we subtract **30000** from it, we receive **8543** kms, which is less than 10000 kms.  After all the commands, we print our current collection of cars with their current mileage and current fuel. | |
| **Input** | **Output** |
| 4  Lamborghini Veneno|11111|74  Bugatti Veyron|12345|67  Koenigsegg CCXR|67890|12  Aston Martin Valkryie|99900|50  Drive : Koenigsegg CCXR : 382 : 82  Drive : Aston Martin Valkryie : 99 : 23  Drive : Aston Martin Valkryie : 2 : 1  Refuel : Lamborghini Veneno : 40  Revert : Bugatti Veyron : 2000  Stop | Not enough fuel to make that ride  Aston Martin Valkryie driven for 99 kilometers. 23 liters of fuel consumed.  Aston Martin Valkryie driven for 2 kilometers. 1 liters of fuel consumed.  Time to sell the Aston Martin Valkryie!  Lamborghini Veneno refueled with 1 liters  Bugatti Veyron mileage decreased by 2000 kilometers  Koenigsegg CCXR -> Mileage: 67890 kms, Fuel in the tank: 12 lt.  Lamborghini Veneno -> Mileage: 11111 kms, Fuel in the tank: 75 lt.  Bugatti Veyron -> Mileage: 10345 kms, Fuel in the tank: 67 lt. |

|  |  |
| --- | --- |
| **JavaScript Input** | **Output** |
| [  '3',  'Audi A6|38000|62',  'Mercedes CLS|11000|35',  'Volkswagen Passat CC|45678|5',  'Drive : Audi A6 : 543 : 47',  'Drive : Mercedes CLS : 94 : 11',  'Drive : Volkswagen Passat CC : 69 : 8',  'Refuel : Audi A6 : 50',  'Revert : Mercedes CLS : 500',  'Revert : Audi A6 : 30000',  'Stop'  ] | Audi A6 driven for 543 kilometers. 47 liters of fuel consumed.  Mercedes CLS driven for 94 kilometers. 11 liters of fuel consumed.  Not enough fuel to make that ride  Audi A6 refueled with 50 liters  Mercedes CLS mileage decreased by 500 kilometers  Volkswagen Passat CC -> Mileage: 45678 kms, Fuel in the tank: 5 lt.  Mercedes CLS -> Mileage: 10594 kms, Fuel in the tank: 24 lt.  Audi A6 -> Mileage: 10000 kms, Fuel in the tank: 65 lt. |
| **Comments** | |
| After we receive the cars with their mileage and fuel, we start driving them. When we get to "**Drive : Volkswagen Passat CC : 69 : 8**" command, our program calculates that there is not enough fuel and we print the appropriate message. Then we refuel the Audi A6 with 50 l of fuel and Revert the Mercedes with 500 kilometers.  When we receive the "Revert : Audi A6 : 30000", we set its mileage to **10000** km, because if the current mileage of the Audi is **38543** kms and if we subtract **30000** from it, we receive **8543** kms, which is less than 10000 kms.  After all the commands, we print our current collection of cars with their current mileage and current fuel. | |
| **JavaScript Input** | **Output** |
| [  '4',  'Lamborghini Veneno|11111|74',  'Bugatti Veyron|12345|67',  'Koenigsegg CCXR|67890|12',  'Aston Martin Valkryie|99900|50',  'Drive : Koenigsegg CCXR : 382 : 82',  'Drive : Aston Martin Valkryie : 99 : 23',  'Drive : Aston Martin Valkryie : 2 : 1',  'Refuel : Lamborghini Veneno : 40',  'Revert : Bugatti Veyron : 2000',  'Stop'  ] | Not enough fuel to make that ride  Aston Martin Valkryie driven for 99 kilometers. 23 liters of fuel consumed.  Aston Martin Valkryie driven for 2 kilometers. 1 liters of fuel consumed.  Time to sell the Aston Martin Valkryie!  Lamborghini Veneno refueled with 1 liters  Bugatti Veyron mileage decreased by 2000 kilometers  Koenigsegg CCXR -> Mileage: 67890 kms, Fuel in the tank: 12 lt.  Lamborghini Veneno -> Mileage: 11111 kms, Fuel in the tank: 75 lt.  Bugatti Veyron -> Mileage: 10345 kms, Fuel in the tank: 67 lt. |

# Programming Fundamentals Final Exam Retake 10.04.2020

## Problem 2. Mirror words

*The SoftUni Spelling Bee competition is here. But it`s not like any other Spelling Bee competition out there, it`s different and a lot more fun! You, of course, are a participant and you are eager to show the competition that you are the best, so go ahead, learn the rules and win!*

On the first line of the input you will be given a **text string**. In order to win the competition you have to find all hidden **word pairs**, read them and mark the ones that are **mirror** **images** of each other.

First of all you have to **extract the hidden word pairs**. Hidden word pairs are:

* Surrounded by "@" or "#" (only one of the two) in the following pattern #wordOne##wordTwo# or @wordOne@@wordTwo@
* At least **3 characters long each** (**without the surrounding symbols**)
* Made up of **letters** **only**

If the second word, **spelled backwards** is the **same** **as the first word** **and vice versa** (**casing matters**!), then they are a **match** and you have to store them somewhere. **Examples** of mirror words:

#Part##traP# @leveL@@Level@ #sAw##wAs#

* If you don`t find any valid pairs print: **"No word pairs found!"**
* If you find valid pairs print their count: **"{valid pairs count} word pairs found!"**
* If there are no mirror words print: **"No mirror words!"**
* If there are mirror words print:

"The mirror words are:

{wordOne} <=> {wordtwo}, {wordOne} <=> {wordtwo}, {wordOne} <=> {wordtwo}, etc…"

### Input / Constraints

* You will recive a string.

### Output

* Print the proper output messages in the proper cases as described in the problem description.
* If there are pairs of mirror words, print them in the end, each pair separated by **", "**.
* Each pair of mirror word must be printed with **" <=> "** between the words.

### Examples

|  |  |
| --- | --- |
| **Input** | |
| @mix#tix3dj#poOl##loOp#wl@@bong&song%4very$long@thong#Part##traP##@@leveL@@Level@##car#rac##tu@pack@@ckap@#rr#sAw##wAs#r#@w1r | |
| **Output** | **Comments** |
| 5 word pairs found!  The mirror words are:  Part <=> traP, leveL <=> Level, sAw <=> wAs | There are 5 green and yellow pairs that meet all requirements and thus are valid.  #poOl##loOp# is valid and looks very much like a mirror words pair but it isn`t because the casings don`t match.  #car#rac# “rac” spelled backwards is "car" but this is not a valid pair because there is only one "#" between the words.  @pack@@ckap@ is also valid but "ckap" backwards is "pakc" which is not the same as "pack", so they are not mirror words. |
| **Input** | |
| #po0l##l0op# @bAc##cAB@ @LM@ML@ #xxxXxx##xxxXxx# @aba@@ababa@ | |
| **Output** | **Comments** |
| 2 word pairs found!  No mirror words! | "xxxXxx" backwards is not the same as "xxxXxx"  @aba@@ababa@ is a valid pair but the word lengths are different, thus these are definitely not mirror words |
| **Input** | |
| #lol#lol# @#God@@doG@# #abC@@Cba# @Xyu@#uyX# | |
| **Output** | **Comments** |
| No word pairs found!  No mirror words! |  |

|  |  |
| --- | --- |
| **JavaScript Input** | |
| [  '@mix#tix3dj#poOl##loOp#wl@@bong&song%4very$long@thong#Part##traP##@@leveL@@Level@##car#rac##tu@pack@@ckap@#rr#sAw##wAs#r#@w1r'  ] | |
| **Output** | **Comments** |
| 5 word pairs found!  The mirror words are:  Part <=> traP, leveL <=> Level, sAw <=> wAs | There are 5 green and yellow pairs that meet all requirements and thus are valid.  #poOl##loOp# is valid and looks very much like a mirror words pair but it isn`t because the casings don`t match.  #car#rac# “rac” spelled backwards is "car" but this is not a valid pair because there is only one "#" between the words.  @pack@@ckap@ is also valid but "ckap" backwards is "pakc" which is not the same as "pack", so they are not mirror words. |
| **JavaScript Input** | |
| [ '#po0l##l0op# @bAc##cAB@ @LM@ML@ #xxxXxx##xxxXxx# @aba@@ababa@' ] | |
| **Output** | **Comments** |
| 2 word pairs found!  No mirror words! | "xxxXxx" backwards is not the same as "xxxXxx"  @aba@@ababa@ is a valid pair but the word lengths are different, thus these are definitely not mirror words |
| **JavaScript Input** | |
| [ '#lol#lol# @#God@@doG@# #abC@@Cba# @Xyu@#uyX#' ] | |
| **Output** | **Comments** |
| No word pairs found!  No mirror words! |  |

# Programming Fundamentals Final Exam Retake 10.04.2020

## Problem 1. Secret Chat

*You have plenty of free time, so you decide to write a program that conceals and reveals your received messages. Go ahead and type it in!*

On the first line of the input you will receive the **concealed message**. After that, until the "Reveal" command is given, **you will be receiving strings** with **instructions** for different **operations** that need to be performed upon the **concealed message** in order to **interpret** **it** and reveal its true content. There are several types of instructions, split by ":|:"

* InsertSpace:|:{index}
  + Inserts a single **empty space** **at the given index**. The given index will always be valid.
* Reverse:|:{substring}
  + If the message contains the given **substring**, **cut it out**, **reverse** it and **add** it at the **end** of the message.
  + If not, print "error".
  + This operation should replace only the first occurrence of the given **substring** **if there are more than one such occurrences**.
* ChangeAll:|:{substring}:|:{replacement}
  + Changes all occurrences of the given substring with the replacement text.

### Input / Constraints

* On the first line, you will receive a string with message.
* On the next lines, you will be receiving commands, split by **":|:"**.

### Output

* After each set of instructions, print the resulting string.
* After the "Reveal" command is received, print this message:  
  "**You have a new text message: {message}**"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| heVVodar!gniV  ChangeAll:|:V:|:l  Reverse:|:!gnil  InsertSpace:|:5  Reveal | hellodar!gnil  hellodarling!  hello darling!  You have a new text message: hello darling! |
| **Comments** | |
| **ChangeAll:|:V:|:l** heVVodar!gniV -> hellodar!gnil (We replace all occurrences of "V" with "l")  **Reverse:|:!gnil**  hellodar!gnil -> !gnil -> ling! -> hellodarling! (We reverse !gnil to ling! And put it in the end of the string)  **InsertSpace:|:5**  hellodarling! -> hello.darling! (We insert a space at index 5)  Finally, after receiving the **"Reveal"** command, we print the resulting message. | |
| **Input** | **Output** |
| Hiware?uiy  ChangeAll:|:i:|:o  Reverse:|:?uoy  Reverse:|:jd  InsertSpace:|:3  InsertSpace:|:7  Reveal | Howare?uoy  Howareyou?  error  How areyou?  How are you?  You have a new text message: How are you? |

|  |  |
| --- | --- |
| **JavaScript Input** | **Output** |
| [  'heVVodar!gniV',  'ChangeAll:|:V:|:l',  'Reverse:|:!gnil',  'InsertSpace:|:5',  'Reveal'  ] | hellodar!gnil  hellodarling!  hello darling!  You have a new text message: hello darling! |
| **Comments** | |
| **ChangeAll:|:V:|:l** heVVodar!gniV -> hellodar!gnil (We replace all occurrences of "V" with "l")  **Reverse:|:!gnil**  hellodar!gnil -> !gnil -> ling! -> hellodarling! (We reverse !gnil to ling! And put it in the end of the string)  **InsertSpace:|:5**  hellodarling! -> hello.darling! (We insert a space at index 5)  Finally, after receiving the **"Reveal"** command, we print the resulting message. | |
| **JavaScript Input** | **Output** |
| [  'Hiware?uiy',  'ChangeAll:|:i:|:o',  'Reverse:|:?uoy',  'Reverse:|:jd',  'InsertSpace:|:3',  'InsertSpace:|:7',  'Reveal'  ] | Howare?uoy  Howareyou?  error  How areyou?  How are you?  You have a new text message: How are you? |