More Exercises: Objects, Classes, Files and Exceptions

Problems for exercises and homework for the “Programming Fundamentals” course @ SoftUni.

Check your solutions here.

I. Objects and Classes

Problem 1. Order by Age

You will receive an **unknown** number of lines. On each line, you will receive array with **3** elements. **The first** element will be string and represents the name of the person. **The second** element will be a **string** and will represent the **ID** of the person. **The last** element will be an **integer** and represents the **age** of the person.

When you receive the command “**End**”, stop taking input and print **all the people**, **ordered** by **age**.

Examples

Input	Output
Georgi 123456 20 Pesho 78911 15 Stefan 524244 10 End	Stefan with ID: 524244 is 10 years old. Pesho with ID: 78911 is 15 years old. Georgi with ID: 123456 is 20 years old.

Input	Output
Maria 123456 120 Georgi 31241 50 Denis 41231 23 End	Denis with ID: 41231 is 23 years old. Georgi with ID: 31241 is 50 years old. Maria with ID: 123456 is 120 years old.

Hints

- For C#, you can use **`.OrderBy(…)`** from **`System.Linq`** to specify according to which parameter to order the people.
- For Java, you can do the same with **`.sorted(…)`** from **Stream API**.

Problem 2. Vehicle Catalogue

You have to make a catalogue for vehicles. You will receive two types of vehicle – **car** or **truck**.

Until you receive the command “**End**” you will receive **lines** of **input** in the format:

```
{typeOfVehicle} {model} {color} {horsepower}
```

After the “**End**” command, you will start receiving **models** of **vehicles**. Print for every received vehicle its **data** in the format:

```
Type: {typeOfVehicle}
Model: {modelOfVehicle}
Color: {colorOfVehicle}
Horsepower: {horsepowerOfVehicle}
```

When you receive the command "**Close the Catalogue**", stop receiving input and print the **average horsepower** for the **cars** and for the **trucks** in the format:

```
{typeOfVehicles} have average horsepower of {averageHorsepower}.
```

The **average horsepower** is calculated by **dividing** the **sum** of **horsepower** for **all** vehicles of the type by the **total count** of **vehicles** from the **same type**.

Format the answer to the **2nd decimal point**.

Constraints

- The type of vehicle will always be **car** or **truck**.
- You will not receive the **same model twice**.
- The received horsepower will be integer in the interval **[1…1000]**
- You will receive at most **50** vehicles.
- **Single** whitespace will be used for **separator**.

Examples

Input	Output
truck Man red 200 truck Mercedes blue 300 car Ford green 120 car Ferrari red 550 car Lamborghini orange 570 End Ferrari Ford Man Close the Catalogue	Type: Car Model: Ferrari Color: red Horsepower: 550 Type: Car Model: Ford Color: green Horsepower: 120 Type: Truck Model: Man Color: red Horsepower: 200 Cars have average horsepower of: 413.33. Trucks have average horsepower of: 250.00.

Input	Output
Car Skoda grey 90 car Nissan black 90 car Bugatti blue 1000 End Skoda Close the Catalogue	Type: Car Model: Skoda Color: grey Horsepower: 90 Cars have average horsepower of: 393.33. Trucks have average horsepower of: 0.00.

Problem 3. * Jarvis

Every kid's dream is to have its own personal robot to be their butler and/or slave. Until now, we could not build a fully functional robot, but we can write a program, which simulates what it would be like to build. Let's call him a code name – **Jarvis**.

Our robot will consist of **6** components – **2** arms, **2** legs, **torso** and a **head**. Make **classes** for these components and your robot should have **fields** for **each** of the **components**.

Each component has **different** properties:

- Arms have:
  - Energy consumption **(integer)**
  - Arm reach distance **(integer)**
  - Count of fingers **(integer)**
- Legs have:
  - Energy consumption **(integer)**
  - Strength **(integer)**
  - Speed **(integer)**
- Torso has:
  - Energy consumption **(integer)**
  - Processor size in centimeters **(double)**
  - Housing material **(string)**
- Head has:
  - Energy consumption **(integer)**
  - IQ **(integer)**
  - Skin material **(string)**

On the first line, you will receive the **maximum energy capacity** of the **robot**. **Until** you receive the command "**Assemble!**", you will continuously receive lines with data for **different** components in format:

`{typeOfComponent} {energyConsumption} {property1} {property2}`

The properties will **always** be given in the **same order** as they are described above. If you receive a **component** which is more **energy efficient** than **previous** one – you should **delete** the old component and **replace** it with the **new** one. When **both** of the components **consume more energy** than the one, which you try to **add** → remove the **one**, which is **added first**.

Input

- On the **first** line, you will receive the **maximum energy capacity** of the robot.
- Until you receive the command "**Assemble!**" you will receive components in the format:
  `{typeOfComponent} {energyConsumption} {property1} {property2}`

Output

- If you do **not** have enough **energy efficient** components to **assemble** the robot print:
  "**`We need more power!`**"
- If you do not have enough parts print:
  "**`We need more parts!`**"
- If you **can** build a **robot** with the given **components** print:

```
Jarvis:
#Head:
###Energy consumption: {head's energy consumption}
###IQ: {head's IQ}
###Skin material: {head's skin material}
#Torso:
###Energy consumption: {torso's energy consumption}
###Processor size: {size of the processor}
###Corpus material: {torso's corpus material}
#Arm:
###Energy consumption: {arm's energy consumption}
```

```
        ###Reach: {arm’s reach}
        ###Fingers: {count of fingers}
        #Arm:
        ###Energy consumption: {arm’s energy consumption}
        ###Reach: {arm’s reach}
        ###Fingers: {count of fingers}
        #Leg:
        ###Energy consumption: {head’s energy consumption}
        ###Strength: {leg’s strength}
        ###Speed: {leg’s speed}
        #Leg:
        ###Energy consumption: {head’s energy consumption}
        ###Strength: {leg’s strength}
        ###Speed: {leg’s speed}
```

Print the **legs** and the **feet** ordered by **energy** consumption in **ascending order**.

Constraints

- Jarvis’ energy will be in the interval **[0…9223372036854775807]**
- Components’ energy will be in the interval **[-2147483648…2147483647]**

Examples

Input	Output
1000 Head 500 20 Leather Torso 300 3 Aluminum Leg 150 20 20 Leg 100 30 30 Arm 500 20 30 Leg 80 30 30 Arm 120 20 5 Arm 100 30 4 Head 200 20 Leather Assemble!	Jarvis: #Head: ###Energy consumption: 200 ###IQ: 20 ###Skin material: Leather #Torso: ###Energy consumption: 300 ###Processor size: 3.0 ###Corpus material: Aluminum #Arm: ###Energy consumption: 100 ###Reach: 30 ###Fingers: 4 #Arm: ###Energy consumption: 120 ###Reach: 20 ###Fingers: 5 #Leg: ###Energy consumption: 80 ###Strength: 30 ###Speed: 30 #Leg: ###Energy consumption: 100 ###Strength: 30 ###Speed: 30

Input	Output
5000 Leg 1000 20 30 Arm 500 30 50	We need more parts!

`Arm 500 30 20`
`Arm 500 30 50`
`Arm 300 60 80`
`Torso 700 30 40`
`Leg 200 100 100`
`Assemble!`	

Input	Output

`500`
`Head 500 20 Leather`
`Torso 300 3 Aluminum`
`Leg 150 20 20`
`Leg 100 30 30`
`Arm 500 20 30`
`Leg 80 30 30`
`Arm 120 20 5`
`Arm 100 30 4`
`Head 200 20 Leather`
`Assemble!`
	`We need more power!`

Hints

- You might want to override the **`ToString(…)`** method in some of your classes.

II. Files

For these tasks, you will receive **sample_text.txt file**, which you have to use to make your **exercises**. Just **submit** the **result** of the tasks as plain **text** in the **Judge**.

Problem 4. Punctuation Finder

Read the file, which is in the resource section of the exercise and print all the **punctuation** marks, which you **find** and **separate** them with **comma and a space**. For punctuation marks you can consider only: "**.**", "**,**", "**!**", "**?**" and "**:**".

Submit the **output** in **judge**.

Examples

File Content	Output
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. *More text will be given…*	*,, ,, .,* *Continues…*

Problem 5. Write to File

Read the **same** file, as in the **previous** task, but this time write everything, **except** the **punctuation** marks to a **new** file. Again, consider as punctuations only: "**.**", "**,**", "**!**", "**?**" and "**:**".

Submit the content of the file in **judge.**

Examples

File Content	Output

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. ***More text will be given…***	Lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua ***Continues…***

Problem 6. ** EXCELlent Knowledge

You received excel table named **sample_table.xlsx**. Write a program, which **reads** the table and **prints** all the columns **separated** with single **pipe** (‘**|**’).

Examples

The **first** line of your table should look like this:

Output
ZIP|Sales|Name|Year|Value| ***Continues…***

Hints

- For C#:
  - Add reference to Microsoft Excel Object Library.
  - You can follow this guide for writing the code.
- For Java:
  - You should create Maven project in IntelliJ. You can make it this way.
  - You can find more information about Apache Maven here.
  - After that follow, this guide to read the Excel file.

III. Exceptions

Problem 7. Play Catch

You will receive on the **first** line an **array** of **integers**. After that you will receive **commands**, which should **manipulate** the array:

- “**Replace {index} {element}**” – **Replace** the element at the given **index** with the given **element**.
- “**Print {startIndex} {endIndex}**” – **Print** the elements from the **start** index to the **end** index **inclusive**.
- “**Show {index}**” – **Print** the element at the **index**.

You have the task to **rewrite** the **messages** from the **exceptions** which can be **produced** from your **program**:

- If you receive an **index**, which does **not exist** in the **array** print:
  “`The index does not exist!`”
- If you receive a **variable**, which is of **invalid type**:
  “`The variable is not in the correct format!`”

When you catch **3** exceptions – **stop** the **input** and **print** the **elements** of the array separated with “`, `”.

Examples

Input	Output
`1 2 3 4 5`	`The index does not exist!`

Replace 1 9 Replace 6 3 Show 3 Show pesho Show 6	4 The variable is not in the correct format! The index does not exist! 1, 9, 3, 4, 5

Input	Output
1 2 3 4 5 Replace 3 9 Print 1 4 Print -3 12 Print 1 5 Show 3 Show 12.3 1, 2, 3, 4, 5	2, 3, 9, 5 The index does not exist! The index does not exist! 9 The variable is not in the correct format! 1, 2, 3, 9, 5

Constraints

- The **elements** of the array will be in **integers** in the interval **[-2147483648…2147483647]**
- You will always receive **valid** string for the **first** part of the **command**, but the **parameters** might be **invalid**
- In the "**Print**" command always be true **startIndex <= endIndex**
- You will always **receive** at least **3** exceptions

Problem 8. * Personal Exception

Write your own exception, which is thrown every time a **negative number** is received from the **console**. The **message** of the **exception** should be "**My first exception is awesome!!!**"

Your task is to print every number **greater or equal** to **0.**

If **negative** number is given as input – **catch** the exception and **print exception's** message. **Stop** the program when your Exception is **thrown**.

Examples

Input	Output
1 2 3 -5	1 2 3 My first exception is awesome!!!

Input	Output
1 2 3 -4 5	1 2 3 My first exception is awesome!!!

Hints

- For C#:
  - Make **new** class for the **exception** and choose appropriate name
  - Inherit the class **`System.Exception`**
  - Make one **constructor**, which **inherits** the **base**
  - Pass the **message** to the **base** constructor
  - In the **`Main()`** make while loop and **throw exception**, if the input number is less than 0
  - Catch the exception and print the message. You can access the message with **`Exception.Message`**

- For Java:
  - Make **new** class for the **exception** and choose appropriate name
  - Inherit the class **`java.lang.Exception`**
  - Make one **constructor**, which **inherits** the **super**
  - Pass the **message** to the **super** constructor
  - In the **`main()`** make while loop and **throw exception**, if the input number is **less than 0**
  - Catch the exception and print the message. You can access the message with **`Exception.getMessage()`**

Submit a **.zip** archive with the **main** method and the **exception's class**.