**Exercises: LINQ**

This document defines the **exercise assignments** for the ["CSharp Advanced" course @ Software University](https://softuni.bg/trainings/1633/csharp-advanced-may-2017). Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/).

* **Students by Group**

Print all students from group number 2. Use LINQ. Order the students by **FirstName**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Mills 1  Andrew Gibson 2  Craig Ellis 1  Steven Cole 2  Andrew Carter 2  END | Andrew Gibson  Andrew Carter  Steven Cole |

* **Students by First and Last Name**

Using the same input as above print all students whose first name is before their last name lexicographically. Use LINQ. Print them in order of appearance.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Mills  Andrew Gibson  Craig Ellis  Steven Cole  Andrew Carter  END | Andrew Gibson  Craig Ellis  Andrew Carter |

* **Students by Age**

Write a LINQ function that finds the first name and last name of all students with age between 18 and 24. The query should return the **first name**, **last name** and **age**. Print them in order of appearance.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Mills 24  Andrew Gibson 21  Craig Ellis 19  Steven Cole 35  Andrew Carter 15  END | Sara Mills 24  Andrew Gibson 21  Craig Ellis 19 |

* **Sort Students**

Using the lambda expressions with LINQ syntax sort the students first by **last name** in **ascending** order and then by **first name** in **descending** order.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Gibson  Andrew Gibson  Craig Ellis  Steven Cole  Andrew Ellis  END | Steven Cole  Craig Ellis  Andrew Ellis  Sara Gibson  Andrew Gibson |

* **Filter Students by Email Domain**

Print all students that have email **@gmail.com** in the order of appearance. Use LINQ.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Mills smills@gmail.com  Andrew Gibson agibson@abv.bg  Craig Ellis cellis@cs.edu.gov  Steven Cole themachine@abv.bg  Andrew Carter ac147@gmail.com  END | Sara Mills  Andrew Carter |

* **Filter Students by Phone**

Print all students with phones starting with Sofia’s phone prefix (starting with **02** / **+3592**). Use LINQ.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Mills 02435521  Andrew Gibson 0895223344  Craig Ellis +3592667710  Steven Cole 3242133312  Andrew Carter +001234532  END | Sara Mills  Craig Ellis |

* **Excellent Students**

Print all students that have **at least one mark Excellent (6)**. Use LINQ.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Mills 6 6 6 5  Andrew Gibson 3 4 5 6  Craig Ellis 4 2 3 4  Steven Cole 5 6 5 5  Andrew Carter 5 3 4 2  END | Sara Mills  Andrew Gibson  Steven Cole |

* **Weak Students**

Write a similar program to the previous one to extract the **students with at least 2 marks under or equal to "3"**. Use LINQ.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sara Mills 6 6 6 5  Andrew Gibson 3 4 5 6  Craig Ellis 4 2 3 4  Steven Cole 5 6 5 5  Andrew Carter 5 3 4 2  END | Craig Ellis  Andrew Carter |

* **Students Enrolled in 2014 or 2015**

Using LINQ, extract and print the **Marks** of the students that **enrolled in 2014 or 2015** (the students from 2014 have 14 as their 5-th and 6-th digit in the **FacultyNumber**, those from 2015 have 15).

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 554214 6 6 6 5  653215 3 4 5 6  156212 4 2 3 4  324413 5 6 5 5  134014 5 3 4 2  END | 6 6 6 5  3 4 5 6  5 3 4 2 |

* **Group by Group**

Create a new class named **Person**. It should consists of **properties** : **name** and **group** (String, Integer). Write a program that extracts all persons (students), **grouped by** **GroupName** and then prints them on the console. Print all group names along with the students in each group in acsending order. Use the **group by** query in LINQ. You will be given an input on the console.

**Output format** : **{group} - {name1}, {name2}, {name3}, ...**

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Ivaylo Petrov 10  Stanimir Svilianov 3  Indje Kromidov 3  Irina Balabanova 4  END | 3 - Stanimir Svilianov, Indje Kromidov  4 - Irina Balabanova  10 - Ivaylo Petrov |

* **Students Joined to Specialties**

Create a new class **StudentSpecialty** that holds **specialty name** and **faculty number**. Create a **Student** class that holds **student name** and  **faculty number**. Create a list of **student specialties,** where each specialty corresponds to a certain student (via the faculty number). You will recieve several specialties in the format :

{specialty name} {specialty name} {faculty number}

Until you reach "**Students:**" , you should add specialties to the collection. After you reach "Students:", you should **start reading students** in the format :

{faculty number} {student's first name} {student's second name}

You should add the students untill you recieve "END" command.

Print **all student** names **alphabetically** along with their **faculty number** and **specialty name**. Use the "**join**" LINQ operator.

**Examples**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student Specialties** | | **join** | **Students** | | **→** | **Result (Joined Students with Specialties)** | | |
| **SpecialtyName** | **FacNum** | **FacNum** | **Name** | **Name** | **FacNum** | **Specialty** |
| Web Developer | 203314 | 215314 | Milena Kirova | Asya Manova | 203314 | Web Developer |
| Web Developer | 203114 | 203114 | Stefan Popov | Asya Manova | 203314 | QA Engineer |
| PHP Developer | 203814 | 203314 | Asya Manova | Diana Petrova | 203914 | PHP Developer |
| PHP Developer | 203914 | 203914 | Diana Petrova | Diana Petrova | 203914 | Web Developer |
| QA Engineer | 203314 | 203814 | Ivan Ivanov | Ivan Ivanov | 203814 | PHP Developer |
| Web Developer | 203914 |  |  | Stefan Popov | 203114 | Web Developer |

|  |  |
| --- | --- |
| **Input** | **Output** |
| Web Developer 203314  Web Developer 203114  PHP Developer 203814  PHP Developer 203914  QA Engineer 203314  Web Developer 203914  Students:  215314 Milena Kirova  203114 Stefan Popov  203314 Asya Manova  203914 Diana Petrova  203814 Ivan Ivanov  END | Asya Manova 203314 Web Developer  Asya Manova 203314 QA Engineer  Diana Petrova 203914 PHP Developer  Diana Petrova 203914 Web Developer  Ivan Ivanov 203814 PHP Developer  Stefan Popov 203114 Web Developer |

* **Little John**

**This problem is originally from the PHP Basics Exam (3 May 2015).**

As you probably know Little John is the right hand of the famous English hero - Robin Hood.A little known fact isthat Little John can't handle Math very well. Before Robin Hood left to see Marry Ann, he asked John to **count** his hay of arrowsand send him an **encrypted** message containing thearrow's count**.** The message should be encrypted since it can be intercepted by the Nottingham’s evil Sheriff. Your task is to help Little John before it is too late (0.10 sec).

You are given **4 input** strings(a hay). Those strings **may or may not** contain arrows. The arrows can be of different type as follows:

* ">----->" – a small arrow
* ">>----->" – a medium arrow
* ">>>----->>" – a large arrow

Note that the **body** of each arrow will always be **5 dashes long**. The **difference** between the arrows is in their **tip** and **tail**. The given 3 types are the only ones you should count, the **rest should be ignored** (Robin Hood does not like them). You should start searching the hays **from the largest** arrow type down **to the smallest** arrow type.

After you find the **count** of each arrow type you should **concatenate** them into one number in the order: small,medium, large arrow(even if the arrow count is 0). Then you **convert** the number in **binary** representation, **reverse** it and **concatenate it again** with the initial binary representation of the number. You **convert** the final binary number again **back to decimal**. This is the encrypted message you should send to Robin Hood.

**Input**

The input will be read from the console. The **data** will be received from 4 input **lines containing strings**.

**Output**

The output should be a decimal number, representing the encrypted count of arrows.

**Constraints**

* The input strings will contain any ASCII character.
* Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| >>>----->>abc>>>----->>  >>>----->>  >----->s  >>-----> | 14535  *The count is: 1 small, 1 medium and 3 large arrows*  *113(dec) = 1110001(bin) -> reversed is 1000111(bin)*  *11100011000111(bin) = 14535(dec)* |

* **Office Stuff**

You are given a sequence of **n** companies in format |**<company> - <amount> - <product>|**. Example:

* |SoftUni - 600 - paper|
* |Vivacom - 600 - pen|
* |XS - 20 - chair|
* |Vivacom - 200 - chair|
* |SoftUni - 40 - chair|
* |XS - 40 - chair|
* |SoftUni - 1 - printer|

Write a program that prints **all companies** in **alphabetical** order. For each company print the product type and their aggregated ordered amounts. Order the products by **order of appearance**. **Print** the result in the following format: **<company>: <product>-<amount>, <product>-<amount>,…** For the orders above the output should be:

* SoftUni: paper-600, chair-40, printer-1
* Vivacom: pen-600, chair-200
* XS: chair-60

**Input**

The input comes from the console. At the first line the number **n** stays alone. At the next **n** lines, we have **n** orders in format |**<company> - <amount> - <product>|**.

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 company**. Company lines should be ordered in **alphabetical** **order**. For each company print the **products** ordered by this company in **order** of **appearance**, along with the total amount for the given product. Each line should be in format **<company>: <product>-<amount>, <product>-<amount>, … <product>-<amount>**

**Constraints**

* The **count** of the lines **n** will be in the range [1 … 100].
* The **<company>** and **<product>** will consist only of **Latin characters**, with length of [1 … 20].
* The **<amount>** will be an integer number in the range [1 … 1000].
* Time limit: 0.1 sec. Memory limit: 16 MB.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 7  |SoftUni - 600 - paper|  |Vivacom - 600 - pen|  |XS - 20 - chair|  |Vivacom - 200 - chair|  |SoftUni - 40 - chair|  |XS - 40 - chair|  |SoftUni - 1 - printer| | SoftUni: paper-600, chair-40, printer-1  Vivacom: pen-600, chair-200  XS: chair-60 | 5  |SoftUni - 200 - desk|  |SoftUni - 40 - PC|  |SoftUni - 200 - desk|  |SoftUni - 600 - paper|  |SoftUni - 600 - textbook| | SoftUni: desk-400, PC-40, paper-600, textbook-600 |

* **Export to Excel**

Write a program to create an Excel file like the one below using an external library. Such as [excellibrary](https://code.google.com/p/excellibrary/), [EPPlus](http://epplus.codeplex.com/), etc for C#.

You are given as **input** course data about **1000 students** in a **.txt** file (tab-separated values). Each line in the input holds **ID**, **first name**, **last name**, **email**, **gender**, **student type**, **exam result**, **homework sent**, **homework** **evaluated**, **teamwork score**, **attendances count**, **bonus**.

