**Homework: Functional Programming**

This document defines the homework assignments from the ["HYPERLINK "https://softuni.bg/courses/advanced-csharp"Advanced C#HYPERLINK "https://softuni.bg/courses/advanced-csharp""HYPERLINK "https://softuni.bg/courses/advanced-csharp" Course @ Software University](https://softuni.bg/courses/advanced-csharp). Please submit as homework a single **zip** / **rar** / **7z** archive holding the solutions (source code) of all below described problems. The solutions should be written in C#.

* **Class Student**

Create a class **Student** with properties **FirstName**, **LastName**, **Age**, **FacultyNumber**, **Phone**, **Email**, **Marks** (**IList<int>**), **GroupNumber**. Create a **List<Student>** with sample students. These students will be used for the next few tasks.

* **Students by Group**

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

* **Students by First and Last Name**

Print all students whose first name is before their last name alphabetically. Use a LINQ query.

* **Students by Age**

Write a LINQ query that finds the first name and last name of all students with age between 18 and 24. The query should return only the **first name**, **last name** and **age**.

* **Sort Students**

Using the extension methods **OrderBy()** and **ThenBy()** with lambda expressions sort the students by first name and last name in descending order. Rewrite the same with LINQ query syntax.

* **Filter Students by Email Domain**

Print all students that have email **@abv.bg**. Use LINQ.

* **Filter Students by Phone**

Print all students with phones in Sofia (starting with **02** / **+3592** / **+359 2**). Use LINQ.

* **Excellent Students**

Print all students that have **at least one mark Excellent (6)**. Using LINQ first select them into a new anonymous class that holds **{ FullName + Marks}**.

* **Weak Students**

Write a similar program to the previous one to extract the **students with exactly two marks "2"**. Use extension methods.

* **Students Enrolled in 2014**

Extract and print the **Marks** of the students that **enrolled in 2014** (the students from 2014 have 14 as their 5-th and 6-th digit in the **FacultyNumber**).

* **\* Students by Groups**

Add a **GroupName** property to Student. Write a program that extracts all students **grouped by** **GroupName** and then prints them on the console. Print all group names along with the students in each group. Use the "**group by into**" LINQ operator.

* **\* Students Joined to Specialties**

Create a class **StudentSpecialty** that holds **specialty name** and **faculty number**. Create a list of **student specialties,** where each specialty corresponds to a certain student (via the faculty number). Print all student names alphabetically along with their faculty number and specialty name. Use the "**join**" LINQ operator. Example:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

* **\*\* LINQ to Excel**

Write a C# 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.

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**.

* Create a class **Student** that holds all aforementioned data fields from the file. Add a field **Result** and a method **CalculateResult()** that calculates the total course **result** of a student using the formula ***(exam result + homework sent + homework evaluated + teamwork + attendances + bonus) / 5****.*
* Create a **Student** object for each student from the **.txt** file and store it in some collection. **Filter** only the **online students** and sort them by their **course result**. Print the resulting student collection in an Excel table. Styling the table is not required.



* **\* Text Gravity**

**This problem is originally from the PHP Basics Exam (31 August 2014). You can test your solution** [**here**](https://judge.softuni.bg/Contests/Practice/Index/84)**.**

Write a program that takes as input a **line length** and **text** and formats the text so that it fits inside several rows, each with length equal to the given line length. Once the text is fitted, each character starts dropping as long as there is an empty space below it.

For example, we are given the text "*The Milky Way is the galaxy that contains our star system*" and **line length** of **10**. If we distribute the text characters such that the **text fits in lines with length 10**, the result is:



Text characters start 'falling' **until no whitespace remain under any character**. The resulting text should be printed as an HTML table with each character in **<td></td>** tags.

**Input**

The input will come from the console. It will consist of two lines.

* The first line will hold the line length.
* The second input line will hold a string.

**Output**

The output consists of the HTML table. Everything should be put inside **<table></table>** tags. Each line should be printed in **<tr></tr>** tags. Each character should be printed in **<td></td>** tags (encode the HTML special characters with the **SecurityElement.Escape()** method). Print **space** " " in all empty cells. See the example below.

**Constraints**

* The **line length** will be integer in the range [1 ... 30].
* The **text** will consist [1 … 1000] ASCII characters.

**Example**

|  |
| --- |
| **Input** |
| 10  The Milky Way is the galaxy that contains our star system |
| **Output** |
| <table><tr><td> </td><td> </td><td> </td><td> </td><td>M</td><td> </td><td> </td><td> </td><td> </td><td> </td></tr><tr><td> </td><td>h</td><td>e</td><td> </td><td>i</td><td>i</td><td>l</td><td> </td><td>y</td><td> </td></tr><tr><td>T</td><td>a</td><td>y</td><td>l</td><td>a</td><td>s</td><td>y</td><td>k</td><td>h</td><td>e</td></tr><tr><td>W</td><td>g</td><td>a</td><td>c</td><td>o</td><td>x</td><td>t</td><td>t</td><td>t</td><td>h</td></tr><tr><td>a</td><td>t</td><td>o</td><td>u</td><td>r</td><td>n</td><td>s</td><td>a</td><td>i</td><td>n</td></tr><tr><td>s</td><td>s</td><td>y</td><td>s</td><td>t</td><td>e</td><td>m</td><td>t</td><td>a</td><td>r</td></tr><table> |

* **\* Uppercase Words**

**This problem is originally from the PHP Basics Exam 29 August 2014 Evening. You can test your solution** [**here**](https://judge.softuni.bg/Contests/Practice/Index/84)**.**

Write a program to **reverse the letters of all uppercase words in a text**. In case an uppercase word stays **unchanged** after reversing its letters, then **double each of its letters**. A word is a sequence of Latin letters separated by **non-letter characters** (e.g. punctuation characters or digits). For example, the text "*PHP5 is the latest PHP currently, YES*" consists of the following words: *PHP*, *is*, *the*, *latest*, *PHP*, *currently*, *YES*.

**Input**

The input will be read from the console. It will consist of a variable number of lines, ending with the command "END".

**Output**

The output should hold the **result text**. Ensure you escape correctly the HTML special characters in the output with the **SecurityElement.Escape() method**.

**Constraints**

* The text will be in **ASCII encoding** (texts in Cyrillic, Arabic, Chinese, etc. are not supported).
* Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

**Examples**

|  |
| --- |
| **Input** |
| Companies like  HP, ORACLE and IBM target their platforms for cloud-based environment.  IList<T> implements IEnumerable<T>. GoPHP is a PHP library. |
| **Output** |
| Companies like  PH, ELCARO and MBI target their platforms for cloud-based environment.  IList&lt;TT&gt; implements IEnumerable&lt;TT&gt;. GoPHP is a PPHHPP library. |

|  |
| --- |
| **Input** |
| IBM announced it delivered the first shipment of its Enterprise Cloud system to a U.K.-based managed service provider. PHP5 is the latest PHP currently, YES |
| **Output** |
| MBI announced it delivered the first shipment of its Enterprise Cloud system to a UU.KK.-based managed service provider. PPHHPP5 is the latest PPHHPP currently, SEY |

* **\* Little John**

**This problem is originally from the PHP Basics Exam (3 May 2015). You may check your solution** [**here**](https://judge.softuni.bg/Contests/Practice/Index/84)**.**

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(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 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**

**This problem is from the Java Basics Exam (21 Sept 2014 Evening). You can test your solution** [**here**](https://judge.softuni.bg/Contests/Practice/Index/34)**.**

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 of 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 |