# Exercises: Functional Programming

Problems for exercises and homework for the [["C# Advanced" course @ Software University](https://softuni.bg/courses/csharp-advanced).](https://softuni.bg/courses/csharp-advanced)

You can check your solutions here: <https://judge.softuni.bg/Contests/1473/Functional-Programming-Exercises>

## Action Point

Write a program that reads a collection of **strings** from the console and then **prints** them onto the **console**. Each name should be printed on a **new** **line**. Use **Action<T>**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Gosho Adasha | Pesho  Gosho  Adasha |

## Knights of Honor

Write a program that reads a collection of **names** as **strings** from the **console**, appends "**Sir**" in front of every name and **prints** it back on the **console**. Use **Action<T>**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Gosho Adasha StanleyRoyce | Sir Pesho  Sir Gosho  Sir Adasha  Sir StanleyRoyce |

## Custom Min Function

Write a simple program that reads from the **console** a set of **integers** and **prints** back on the **console** the **smallest** **number** from the collection. Use **Func<T, T>**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 4 3 2 1 7 13 | 1 |

## Find Evens or Odds

You are given a lower and an upper bound for a range of integer numbers. Then a command specifies if you need to list all even or odd numbers in the given range. Use **Predicate<T>**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 10  odd | 1 3 5 7 9 |
| 20 30  even | 20 22 24 26 28 30 |

## Applied Arithmetics

Write a program that executes some mathematical operations on a given collection. On the **first line** you are given **a list of numbers**. On the **next lines** you are passed **different commands** that you need to **apply to all the numbers** in the list:

* **"add"** -> add 1 to each number
* **"multiply"** -> multiply each number by 2
* **"subtract"** -> subtract 1 from each number
* **"print"** -> print the collection
* "**end**" -> ends the input

Use functions.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5  add  add  print  end | 3 4 5 6 7 |
| 5 10  multiply  subtract  print  end | 9 19 |

## Reverse and Exclude

Write a program that reverses a collection and removes elements that are divisible by a given integer **n**. Use predicates/functions.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5 6  2 | 5 3 1 |
| 20 10 40 30 60 50  3 | 50 40 10 20 |

## Predicate for Names

Write a program that filters a list of names according to their length. On the first line, you will be given an integer **n,** representing a name's length. On the second line, you will be given some names as strings separated by space. Write a function that prints only the names whose length is **less than or equal** to **n**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Kurnelia Qnaki Geo Muk Ivan | Geo  Muk  Ivan |
| 4  Karaman Asen Kiril Yordan | Asen |

## Custom Comparator

Write a custom comparator that sorts all even numbers before all the odd ones in ascending order. Pass it to **Array.Sort()** function and print the result. Use functions.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5 6 | 2 4 6 1 3 5 |
| -3 2 | 2 -3 |

## List of Predicates

Find all numbers in the range 1...N that are divisible by the numbers of a given sequence. On the first line, you will be given an integer **N** – which is the end of the range. On the second line, you will be given a sequence of integers which are the dividers. Use predicates/functions.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10  1 1 1 2 | 2 4 6 8 10 |
| 100  2 5 10 20 | 20 40 60 80 100 |

## Predicate Party!

Ivancho’s parents are on a vacation for the holidays and he is planning an epic party at home. Unfortunately, his organizational skills are next to non-existent, so you are given the task to help him with the reservations.

On the **first line,** you receive a **list** **with all the people** that are coming. On the **next lines**, until you get the **"Party!" command**, you may be asked to **double** or **remove** **all the people** that apply to a given **criteria**. There are **three different** **criteria**:

* Everyone that has his **name** **starting** with a **given string**
* Everyone that has a **name** **ending** with a **given string**
* Everyone that has a **name** with a **given length**.

Finally, **print all the guests** who are going to the party **separated by** "," and then **add the ending** "are going to the party!". If there are **no guests** going to the party print "Nobody is going to the party!". See the examples below:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Misho Stefan  Remove StartsWith P  Double Length 5  Party! | Misho, Misho, Stefan are going to the party! |
| Pesho  Double StartsWith Pesh  Double EndsWith esho  Party! | Pesho, Pesho, Pesho, Pesho are going to the party! |
| Pesho  Remove StartsWith P  Party! | Nobody is going to the party! |

## Party Reservation Filter Module

You need to implement a filtering module to a party reservation software. First, to the Party Reservation Filter Module (PRFM for short) is **passed a list** with invitations. Next the PRFM receives a **sequence of commands** that specify whether you need to add or remove a given filter.

Each PRFM command is in the given format:

"**{command;filter type;filter parameter}"**

You can receive the following PRFM commands:

* "**Add filter**"
* "**Remove filter**"
* "**Print**"

The possible PRFM filter types are:

* "**Starts with**"
* "**Ends with**"
* "**Length**"
* "**Contains**"

All PRFM filter parameters will be a string (or an integer only for the "**Length"** filter). Each command will be valid e.g. you won’t be asked to remove a non-existent filter. The input will **end** with a "**Print**" command, after which you should print all the party-goers that are left after the filtration. See the examples below:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Misho Slav  Add filter;Starts with;P  Add filter;Starts with;M  Print | Slav |
| Pesho Misho Jica  Add filter;Starts with;P  Add filter;Starts with;M  Remove filter;Starts with;M  Print | Misho Jica |

## Inferno III

On the **first line** you are given **a sequence of numbers**. Each number is a gem and the **value** represents its **power**. On the next lines, until you receive the "**Forge**" command, you will be receiving commands in the following format: "**{command;filter type;filter parameter}"**

**Commands** can be:

* "**Exclude**" - marks a gem for **exclusion** from the set if it meets a **given condition**
* "**Reverse**" - **deletes** a previous **exclusion**
* "**Forge**" – end of input

The possible filter types are:

* "**Sum Left**" - tests if a gem’s **power** **added** to the gem standing to **its** **left** gives a **certain value**
* "**Sum Right**" - tests if a gem’s **power** **added** to the gem standing to **its** **right** gives a **certain value**
* "**Sum Left** **Right**" - sums the gems' power with **both** its **left** and **right** neighbors and if a gem has **no neighbor** to its right or to its left (first or last element), **add 0** to the gem

All filter **parameters** will be **integers**. Note that **changes** to the sequence **are applied** only **after forging**. This means that the gems are fixed at their positions and **every function** occurs on the **original set**, so every gems power is considered, no matter if it is marked to be excluded or not. To better understand the problem, see the examples below:

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 3 4 5  Exclude;Sum Left;1  Exclude;Sum Left Right;9  Forge | 2 4 | 1. Marks for exclusion all gems for which the sum with neighbors to their left equals 1, e.g. 0 + **1** = 1  2. Marks for exclusion all gems for which the sum with neighbors to their left and their right equals 9, e.g.  2 + **3** + 4 = 9  4 + **5** + 0 = 9 |
| 1 2 3 4 5  Exclude;Sum Left;1  Reverse;Sum Left;1  Forge | 1 2 3 4 5 | 1. Marks for exclusion all gems for which the sum with their gem peers to the left equals 1, e.g. 0 + 1 = 1  2. Reverses the previous exclusion. |

## TriFunction

Write a program that traverses a collection of names and returns the **first name,** whose sum of characters is **equal** to or **larger** than a given number **N,** which will be given on the first line. Use a function that **accepts another function** as one of its parameters. Start off by building a regular function to hold the basic logic of the program. Something along the lines of **Func<string, int, bool>**. Afterwards create your main function which should accept the first function as one of its parameters.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 800  Qvor Qnaki Petromir Sadam | Petromir |