# Exercise: Generics

Problems with exercise and homework for the ["CSharp Advanced" course @ Software University.](https://softuni.bg/trainings/3584/csharp-advanced-january-2022)

You can check your solutions in [Judge](https://judge.softuni.org/Contests/1475/Generics-Exercise).

## Generic Box of String

Create a generic **class** **Box** that can be initialized with **any type** and **store the value**. Override the **ToString()** method and **print** the **type** and **stored** **value**.

### Input

* On the first line, you will get **n** - the number of strings to read from the console.
* On the next **n** lines, you will get the **actual strings**.
  + For each of them, create a box and call its **ToString()** method to **print** its data on the console.

**Output**

* The output should be in the given format:

**"{class full name: value}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  life in a box  box in a life | System.String: life in a box  System.String: box in a life |
| 3  Peter  Simon  Griffin | System.String: Peter  System.String: Simon  System.String: Griffin |

## Generic Box of Integer

Use the description of the previous problem but now, test your generic box with Integers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  1001 | System.Int32: 1001 |
| 3  7  123  42 | System.Int32: 7  System.Int32: 123  System.Int32: 42 |

## Generic Swap Method Strings

Create a **generic** **method** that receives **a list**, containing **any** **type** of **data** and **swaps** **the elements** at **two** **given** **indexes**.

### Input

* On the first line, you will read **n** number of boxes of type **string** and **add** them to the list.
* On the next line, however, you will receive a **swap command** consisting of **two indexes**.

### Output

* Use the **method** you've created to swap the elements that correspond to the given indexes and **print each element in the list**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Peter  George  Swap me with Peter  0 2 | System.String: Swap me with Peter  System.String: George  System.String: Peter |
| 2  SoftUni  Hello  0 1 | System.String: Hello  System.String: SoftUni |

## Generic Swap Method Integers

Use the description of the previous problem, but now, **test** your list of generic boxes with **integers**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  7  123  42  0 2 | System.Int32: 42  System.Int32: 123  System.Int32: 7 |
| 3  1  2  3  0 1 | System.Int32: 2  System.Int32: 1  System.Int32: 3 |

## Generic Count Method Strings

Create a **method** that receives as an argument a **list of any type, that can be compared** and an **element of the given type**. The method should **return the count of elements that are greater than the value of the given element**. **Modify your Box class** to support **comparison by the value** of the stored data.

### Input

* On the **first** line, you will receive **n** - the number of **elements** to **add to the list**.
* On the next **n** lines, you will receive the **actual elements**.
* On the **last** line, you will get the **value** of the **element** for comparison.

### Output

* Print the **count of elements** that are **larger** than the value of the given element.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  aa  aaa  bb  aa | 2 |
| 1  aaa  aa | 1 |

## Generic Count Method Doubles

Use the description of the previous problem, but now, **test** your **list** of generic boxes with **doubles**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  7.13  123.22  42.78  7.55 | 2 |
| 3  1.1  2.3  3.2  1.5 | 2 |

## Tuple

A [**Tuple**](https://msdn.microsoft.com/en-us/library/system.tuple(v=vs.110).aspx) is a class in C#, in which you can store a few objects. First, we are going to focus on the **Tuple's type**, which contains two objects. The first one is "**item1**" and the second one is "**item2**". It is kind of like a **KeyValuePair**, except – it **simply has items,** which are **neither key nor value**. Your task is to create a class "**Tuple**", which holds two objects. The first one will be "**item1**" and the second one – "**item2**". The tricky part here is to make the class **hold generics**. This means, that when you create a new object of class – "**Tuple**", there should be a way to explicitly specify both items' **types** **separately**.

### Input

The input consists of **three** lines:

* The **first** one is holding a **person's name** and **an address**. They are separated by space(s). Your task is to **collect** them in the **tuple** and **print** them on the **console**. Format of the input:

**{first name} {last name} {address}**

* The second line holds a **name** of a personand the **amount of beer** (int) he can drink. Format:

**{name} {liters of beer}**

* The last line will hold an **integer** and a **double**. Format:

**{integer} {double}**

### Output

* Print the tuples’ items in format: **{item1} -> {item2}**

### Constraints

* Use the good practices we have learned. Create the class and make it have getters and setters for its class variables. The input will be **valid**, no need to check it explicitly!

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Adam Smith California  Mark 2  23 21.23212321 | Adam Smith -> California  Mark -> 2   1. -> 21.23212321 |
| William Donovan York  Richard 2999999  10 10 | William Donovan -> York  Richard -> 2999999  10 -> 10 |

## Threeuple

Create a Class **Threeuple**. Its name is telling us, that it will hold no longer, just a pair of objects. The task is simple, our **Threeuple** should **hold three objects**. Make it have getters and setters. You can even extend the previous class

### Input

The input consists of three lines:

* The first one is holding a name, an address and a town. Format of the input:

**{first name} {last name} {address} {town}**

* The second line is holding a **name**, **beer** **liters**, and a **boolean** variable with value - **drunk** or **not**. Format:

**{name} {liters of beer} {drunk or not}**

* The last line will hold a **name**, a **bank** **balance** (**double**) and a **bank name**. Format:

**{name} {account balance} {bank name}**

### Output

* Print the Threeuples' objects in format:

**"{firstElement} -> {secondElement} -> {thirdElement}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Adam Smith Wallstreet New York  Mark 18 drunk  Karren 0.10 USBank | Adam Smith -> Wallstreet -> New York  Mark -> 18 -> True  Karren -> 0.1 -> USBank |
| Anatoly Andreevich Kutuzova Kaliningrad  Marley 9 not  Grant 2 NGB | Anatoly Andreevich -> Kutuzova -> Kaliningrad  Marley -> 9 -> False  Grant -> 2 -> NGB |

**Note**: You may extend your previous solution.

## \*Custom Linked List

Now you have the needed knowledge to extend the custom linked list you have created during the previous workshop and your task is to make it **generic**. Upload your solution without the bin and obj folders in Judge.