LAB 08 - Generics

Problem 0. Generic Box

Create a generic class Box that can be initialized with **any** type and **store** the value. **Override** the **ToString()** method to print the type and the value of the data stored in the format **{class full name: value}.**

Note

This problem does not have tests in Judge but instead, the class is used in the next problems.
In order to get a class' full name, use **`.GetType().FullName`** property. You can read more here.

Examples

Input	Output
`123123`	`System.Int32: 123123`
`life in a box`	`System.String: life in a box`

Problem 1. Generic Box of String

Use the class that you've created for the previous problem and test it with the class **System.String.** 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.

Examples

Input	Output

`2`
`life in a box`
`box in a life`
	`System.String: life in a box`
	`System.String: box in a life`

Problem 2. Generic Box of Integer

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

Examples

Input	Output

`3`
`7`
`123`
`42`
	`System.Int32: 7`
	`System.Int32: 123`
	`System.Int32: 42`

Problem 3. 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.
As in the previous problems 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. 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`
`Pesho`
	`System.String: Swap me with Pesho`
	`System.String: Gosho`

Gosho Swap me with Pesho 0 2	System.String: Pesho

Problem 4. 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

Problem 5. Generic Count Method Strings

Create a **method** that receives as 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 **comparing by value** of the data stored.
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 to which you need to compare every element in the list.

Examples

Input	Output
3 aa aaa bb aa	2

Problem 6. 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

Problem 7. Custom List

Create a generic data structure that can store **any** type that **can** be compared. Implement functions:

- **void Add(T element)**
- **T Remove(int index)**
- **bool Contains(T element)**
- **void Swap(int index1, int index2)**
- **int CountGreaterThan(T element)**
- **T Max()**

- **T Min()**

Create a command interpreter that reads commands and modifies the custom list that you have created. Set the custom list's type to string. Implement the commands:

- **Add <element>** - Adds the given element to the end of the list
- **Remove <index>** - Removes the element at the given index
- **Contains <element>** - Prints if the list contains the given element **(True or False)**
- **Swap <index> <index>** - Swaps the elements at the given indexes
- **Greater <element>** - Counts the elements that are greater than the given element and prints their count
- **Max** - Prints the maximum element in the list
- **Min** - Prints the minimum element in the list
- **Print** - Prints all elements in the list, each on a separate line
- **END** - stops the reading of commands

There will **not** be any **invalid** input commands.

Examples

Input	Output

`Add aa`
`Add bb`
`Add cc`
`Max`
`Min`
`Greater aa`
`Swap 0 2`
`Contains aa`
`Print`
`END`
	`cc`
	`aa`
	`2`
	`True`
	`cc`
	`bb`
	`aa2`

Problem 8. Custom List Sorter

Extend the previous problem by creating an additional **Sorter class**. It should have a single static **method Sort()** which can sort objects of type **CustomList** containing any type that can be compared. **Extend the command list** to support one additional command Sort:

- **Sort** - Sort the elements in the list in ascending order.

Examples

Input	Output

`Add cc`
`Add bb`
`Add aa`
`Sort`
`Print`
`END`
	`aa`
	`bb`
	`cc`

Problem 9. *Custom List Iterator

For the print command, you have probably used a `for` loop. Extend your custom list class by making it implement **IEnumerable<T>.** This should allow you to iterate your list in a foreach statement.

Examples

Input	Output

`Add aa`
`Add bb`
	`cc`
	`aa`

Add cc Max Min Greater aa Swap 0 2 Print END	2 cc bb aa

Problem 10. **Tuple**

There is something, really annoying in C#. It is called a Tuple. It is a class, which may store a few objects but let's focus on the type of Tuple 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**. The annoyance is coming from the fact, that you have no idea what these objects are holding. The class name is telling you nothing, the methods which it has – also. So, let's say for some reason we would like to try to implement it by ourselves.

The task: Create a class "**Tuple**", which is holding two objects. Like we said, 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 the items type separately.

Input

The input consists of three lines:

- The first one is holding a person 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 person and 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!

Example

Input	Output
Sofka Tripova Stolipinovo Az 2 23 21.23212321	Sofka Tripova -> Stolipinovo Az -> 2 23 -> 21.23212321