# Advanced Exercises: Haskell playground

## Descending Sort Array

Write a Haskell function which receives an array and returns it ordered in descending order. (you will need this one for the next tasks).

### Examples

|  |  |
| --- | --- |
| **Function parameter** | **Function result** |
| [25, 43, 11, 28, 44] | [11, 25, 28, 43, 44] |
| ["asd", "bsd", "csd"] | ["csd", "bsd", "asd"] |

### Note

Due to the function receiving **different types** of lists, it can **vary** in **definitions**.

## Lambada Expressions

IT Researchers at the MIT (Massachusetts Institute of Technology) have discovered that, at certain (room) temperatures, the lambda expressions in their code are acting strange. In particular, a dancing behavior has been observed in them. The specialists choose to call this phenomenon – The Lambada expressions.

### Input

You will be given several input lines in the following format:

{selector} => {selectorObject}.{property}

All elements of the input are **strings**. You need to store every **lambada expression**, as it is **given**.

In some rare cases, you will receive the input “**dance**”, which indicates that the lambada expressions are starting to dance. When that happens, you must **COPY** the **selectorObject ONCE** with a **dot** (“**.**”) **between** it., so that it becomes “{selectorObject}.{selectorObject}”

in **every** lambada expression’s **condition**.   
In other words, if you have “x => x.Value”, and you say “**dance**”, you should get “x => x.x.Value”.

Check the examples for more info.

If you meet a **selectorObject** which **already exists**, you must **replace** its **property** with the **given new one**.

When you receive the command “lambada”, you must **stop** reading input.

### Output

Print all **lambada expressions** in **order of insertion**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| x => x.Key  object => object.Property  entry => entry.Name  lambada | x => x.Key  object => object.Property  entry => entry.Name |
| x => x.Key  x => x.Value  x => x.Name  dance  dance  dance  lambada | x => x.x.x.x.Name |

## LINQuistics

LINQ is the greatest .NET component of all time. You can do almost anything with it. That’s why you have been tasked to do almost everything you can with it.

### Input

You will be given several input lines containing information about collections, and LINQ methods that have been called on them, in the following format:

{collection}.{method1}().{method2}()....{methodN}()

The **count** of **methods** may **vary**. Your task is to **store** every collection and the **methods** that have been executed on it. If the collection **already exists**, you must **ADD** the new methods to it. **Duplicate** methods should be **REMOVED**.

### Output

If you are given **only** a **collection name**, you must **print** the **methods** that have been **executed** on the collection, **ordered** by their **length** in **descending order**. If **2 methods** have the **same length**, **order** them by the **count** of **unique symbols** they have in **their names** in **descending order**.

Each method must be printed on a **new line**, with a prefixed **asterisk and space** (“\* ”).

If the collection name does **NOT** exist, you should **IGNORE** that line of input.

If you are given **only** a **digit**, you must **take** **the collection** which **has the most methods**, and print the **first N methods**, with the **lowest length** (**N** being the **digit** **given** in the **input**). If there are **less** than **N** methods you must print **all** of them in the **same order**.

**NOTE**: When printing, you must print **only** the **method name**, **without** its brackets (e.g. “First”, **not** “First()”).

The input sequence ends when you receive the command “**exit**”. After the ending command, you will receive one last line in the following format:  
{method} {selection}

You must **take all** collections, which **CONTAIN** the **given method**, and print them. The selection will either be “collection” or “all”.   
If you have “collection”, you must **print** only the **collections’ names** in the final output.  
If you have “all”, you must **print** the **collections** **and their methods** in the following format:

|  |
| --- |
| {collection}  \* {method1}  \* {method2} ... |

The collections must be printed ordered by the **count of their methods** in **descending order**.

If **2 collections** have the **same** **amount of methods**, print the **one** whose **shortest method name** is **longer** than the other one’s **shortest method name**.

The **methods** must be printed, ordered by their **length** in **descending order**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| participants.Max().Reverse().ThenBy()  participants.OrderBy.Select()  participants  participants.ToDictionary()  collection.Max()  collection.Break()  exit  Max all | \* OrderBy  \* Reverse  \* ThenBy  \* Select  \* Max  participants  \* ToDictionary  \* Reverse  \* OrderBy  \* ThenBy  \* Select  \* Max  collection  \* Break  \* Max |
| elements.Sort()  elements.OrderBy()  bound  elements.Reverse().Select().ThenBy()  keys.Reverse().OrderByDescending()  keys.Reverse().ThenByDescending()  3  keys.Reverse().OrderBy().ThenBy()  values.ToString().ToString().ThenBy()  exit  Reverse collection | \* Sort  \* Select  \* ThenBy  keys  elements |