High-Quality Code Exam – Phonebook

A system designed to host a **phonebook** holds phonebook entries (**name** and **list of phone numbers**). The system allows **adding to the phonebook**, **changing phones** and **listing the entries with paging**.

Phone numbers can be given in any standard form, e.g. "(02) 981 22 33", "123", "(+1) 123 456 789", "0883 / 456-789", "0888 88 99 00", "888-88-99-00", "+359 (888) 41-80-12", "00359 (888) 41-80-12" or "+359527734522" and should be stored in the system in their canonical form. The canonical form for given phone number is obtained just like in the traditional GSM devices. First all non-digit characters except "+" are removed. After that, if the phone starts with "00", the "00" is replaced with "+". After that, any leading zeroes are removed. Finally the default country code "+359" is inserted in the beginning of the phone number if no country code is available (if the phone does not start with "+").

You are assigned to create a program that executes a sequence of commands against the phonebook. Each command consists of a single text line and produces zero, one or more text lines. The commands are described below:

- **AddPhone(name, phone1, phone2, …)** – adds a new entry to the phone book. The entry should specify **name** and **list of phones** (at least 1 and at most 10). The names in the phonebook are unique (duplicates are not accepted) and case-insensitive. Adding phones for same name always performs **merging**: only the non-repeating canonical phones enter in the list of phones. The command outputs “**Phone entry created**” as a result when the name is missing in the phonebook and "**Phone entry merged**" when the name already exists in the phonebook.
- **ChangePhone(oldNumber, newNumber)** – changes the specified old phone number in all phonebook entries with a new one. The command prints “**X numbers changed**” as a result, where **X** is the number of the changed old phone numbers in the system. Changing a phone number works with **merging** and thus any duplicating phone numbers are omitted.
- **List(startIndex, count)** – lists the phonebook entries with paging. The page is specified by **start index** (zero-based) and **count** in the phonebook assuming that the entries are sorted by name (case-insensitive). The **count** specifies the page size (the number of phonebook entries to be retrieved). The listed phonebook entries should be printed in the following form: "**[name: phone1, phone2, …]**", each on a separate line. The name should appear in the same casing as when it was first added to the phonebook. The phone numbers should be sorted alphabetically (as text). In case the start index is invalid or the count is invalid, the command prints "**Invalid range**".
- **End** – indicates the end of the input sequence of commands. "**End**" stops the commands processing without any output. It is always the last command in the input sequence.

Input

The input data comes from the console. It consists of a sequence of commands, each staying at a separate single line. The input ends by the "**End**" command.

The **input data will always be valid** and in the described format. There is no need to check it explicitly.

Output

The output should be on the console. It should consist of the outputs from each of the commands from the input sequence.

Constraints

- The **name** will be non-empty English text (less than 200 characters) and cannot contain "**,**", "**:**" and "**\n**", as well as leading or trailing whitespace. Names are case-insensitive, e.g. "*Peter*" is the same as "*peter*" and "PETER".
- The **phone numbers** will contain only digits, whitespace and the special characters "**+**", "**-**", "**/**", "**(**" and "**)**". Phone numbers cannot contain "**,**" and "**\n**", leading or trailing whitespace. Phone numbers always have of [3..50] digits. Phone numbers could have at most one leading zero.
- The **start** will be integer number in the range [0…1000000].
- The **count** will always be integer number in the range [1…20].
- There is a single space after each "**,**" in the input and no space before it. There are no spaces around the "**(**" and "**)**" in the commands.
- The input cannot exceed **2 MB**.
- Allowed working time for your program: 2 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example
`AddPhone(Kalina, 0899 777 235, 02 / 981 11 11)`	`Phone entry created`
`AddPhone(kalina, +359899777235)`	`Phone entry merged`
`AddPhone(KALINA, (+359) 899777236)`	`Phone entry merged`
`AddPhone(Kalina (new), 0899 76 15 33)`	`Phone entry created`
`List(0, 1)`	`[Kalina: +35929811111, +359899777235, +359899777236]`
`List(10, 10)`	`Invalid range`
`ChangePhone((+359) 899 777236, 0883 22 33 44)`	`1 numbers changed`
`AddPhone(Zhoro.Telerik, 0893 656 756)`	`Phone entry created`
`AddPhone(Alex St.Zagora, 042 77 33 55)`	`Phone entry created`
`ChangePhone(0893 656 756, 0898 123 456)`	`1 numbers changed`
`List(1, 3)`	`[Kalina: +35929811111, +359883223344, +359899777235]`
`ChangePhone(042 77 33 55, 02 98 11 111)`	`[Kalina (new): +359899761533]`
`ChangePhone((02) 9811111, 088 322 33 44)`	`[Zhoro.Telerik: +359898123456]`
`List(0, 2)`	`1 numbers changed`
`ChangePhone((02) 9811111, 088 322 33 44)`	`2 numbers changed`
`End`	`[Alex St.Zagora: +359883223344]`
	`[Kalina: +359883223344, +359899777235]`
	`0 numbers changed`

Problem 1. Code Refactoring

Refactor the source code to improve its quality following the best practices introduced in the course “High-Quality Programming Code”. Find 4 common refactoring problems.

12 points

Problem 2. StyleCop

Fix all **StyleCop** issues you can find**.**

8 points

Problem 3. Design Patterns

Add these design patterns to the solution - Command, Factory.

18 points

Problem 4. Unit Testing

Design and implement **unit tests for all public methods from the `IPhonebookRepository` interface**. Any other code is not required to be tested. The **code coverage** should be at **least 85% for the class implementing `IPhonebookRepository`**. Be sure to test all major execution scenarios + all interesting border cases and special cases. Use VSTT and VS code coverage.

12 points

Problem 5. Code Documentation

Document all public methods and properties defined in **the `IPhonebookRepository` interface** using XML documentation. Any other documentation is not required.

7 points

Problem 6. Performance Bottlenecks

Find any **performance bottlenecks** and briefly describe them with a comment in the code.

Fix the problems if possible. In this case leave the comments describing the bottlenecks.

8 points

Problem 7. Bug Fixing

Debug the code and **find and fix the bugs** in it.

10 points

Problem 8. New Feature

Implement deleting command allowing you to safely remove a phone number from the phonebook. It should have the following signature – **Remove(phoneNumber)** where **phoneNumber** is canonical phone number described above. If the phone number is invalid, the program should show "**Invalid phone number**" message. If the phone number cannot be found, the program should show "**Phone number could not be found**" message. If the user associated with the removed phone numbers does not have any other phone numbers listed, the user should be removed as well.

10 points

Problem 8. SOLID

Add the five solid principle to all implementations of IPhonebookRepository.

15 points

Ensure the program works correctly after all modifications!