# Homework: AVL Trees and AA Trees

This document defines the **homework assignments** for the ["Data Structures" course @ Software University](https://softuni.bg/trainings/1857/data-structures-january-2018)

. You can submit your code in the SoftUni Judge System - <https://judge.softuni.bg/Contests/630/AVL-Trees-and-AA-Trees-CSharp-Exercise>.

## First / Last List

You need to implement a **“first / last list” data structure** that performs fast enough the following operations:

* **Add(*element*)**
  + **Adds an element** to the structure (duplicate elements are accepted)
* **Count**
  + Returns the **number of elements** in the structure
* **First(*count*)**
  + Returns the **earliest added *count* elements** (in the order of addition from first to last)
  + Throws an **exception** in case the structure holds less than *count* elements
* **Last(*count*)**
  + Returns the **latest added *count* elements** (in the order of addition from last to first)
  + Throws an **exception** in case the structure holds less than *count* elements
* **Min(*count*)**
  + Returns the **smallest *count* elements** (ordered from the smallest to the largest as first criteria and by the order of adding as second criteria)
  + Throws an **exception** in case the structure holds less than *count* elements
  + Note: the elements should be **comparable** in order to find the minimal element
* **Max(*count*)**
  + Returns the **biggest *count* elements** (ordered from the biggest to the smallest as first criteria and by the order of adding as second criteria)
  + Throws an **exception** in case the structure holds less than *count* elements
  + Note: the elements should be **comparable** in order to find the maximal element
* **Remove-All(*element*)**
  + **Removes all elements** that are equal to the passed element
  + Returns the **number of removed elements** or 0 when no elements are matched
  + Note: the elements should be **comparable** in order to find which pairs of elements are equal
* **Clear()**
  + **Removes all elements** from the data structure

### Input and Output

You are given a **Visual Studio C# project skeleton** (unfinished project) / **Eclipse Java project** / **Visual Studio C++ project** holding the interface IFirstLastList, the unfinished class FirstLastList and **tests** covering its **functionality** and its **performance**.

Your task is to **finish this class** to make the tests run correctly.

* You are **not allowed to change the tests**.
* You are **not allowed to change the interface**.

### Interface IFirstLastList

The interface IFirstLastList in C# looks like the code below:

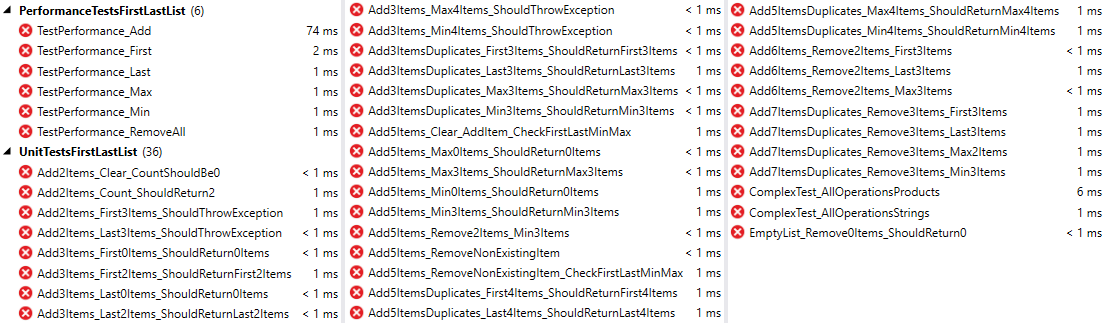
|  |
| --- |
| public interface IFirstLastList<T> where T : IComparable<T>  {  void Add(T element);  int Count { get; }  IEnumerable<T> First(int count);  IEnumerable<T> Last(int count);  IEnumerable<T> Min(int count);  IEnumerable<T> Max(int count);  void Clear();  int RemoveAll(T element);  } |

The interface IFirstLastList in Java looks like the code below:

|  |
| --- |
| **public** **interface** IFirstLastList<T **extends** Comparable<T>> {  **void** add(T element);  **int** getCount();  Iterable<T> first(**int** count);  Iterable<T> last(**int** count);  Iterable<T> min(**int** count);  Iterable<T> max(**int** count);  **void** clear();  **int** removeAll(T element);  } |

### Unit Tests

The C# project holds many automated **tests** for the data structure **functionality** and its **performance**:



## \* Implement AVL Tree Deletion

Extend your AVL Tree to support:

* void DeleteMin() 🡪 deletes the minimum element (balances the tree if necessary)
* void Delete(T item) 🡪 deletes the given element (balances the tree if necessary)

You are given a skeleton with additional tests that cover delete operations.

# Exercises: Ropes and Tries

This document defines the **exercise assignments** for the ["Data Structures" course @ Software University](https://softuni.bg/trainings/1857/data-structures-january-2018). You can submit your code in the SoftUni Judge System - <https://judge.softuni.bg/Contests/639/Ropes-and-Tries-CSharp-Exercise>.

## Text Editor

Your task is to implement a **text editor**. The program will be used by **multiple users** at a time. Each user will have its own **text**, which he should be able to **edit**. Also, we should be able to see **all of the users** that are using our application. You will receive commands, which you need to execute:

* login {username} – the given user can start executing commands
* logout {username} – the given user can no longer edit his string
* {username} {command} {parameters} – this is the pattern we will be using for the following **user commands**:
  + insert {index} {string} – inserts the given string in the given position
  + prepend {string} – inserts the given string in the beginning of the string
  + substring {start index} {length} – replaces the user string with a substring from it
  + delete {start index} {length} – removes part of the user string
  + clear – deletes the user string
  + length – returns the length of the user string
  + print – returns the user string
  + undo – reverts the last operations on the user string. Can be used multiple times
* users – returns all users currently logged in
* users {prefix} – returns all users starting with the given prefix
* end – stops the program

You will be given **skeleton** containing **interface** with the desired functionality. User commands should be **ignored** if the user is **not logged** in. If the same user tries to login twice, delete his old string. Choose the appropriate data structures, which will handle big input size. Command parameters will be split with exactly 1 **whitespace**. All commands that lead to Exceptions, should be ignored.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| login pesho  pesho prepend "hello"  pesho print  pesho length  end | hello  5 |
| **Input** | **Output** |
| login pesho  pesho prepend "hello"  pesho insert 0 "H"  pesho print  pesho delete 1 1  pesho print  pesho insert 5 ", my name is pesho"  pesho print  pesho substring 0 5  pesho print  end | Hhello  Hello  Hello, my name is pesho  Hello |

|  |  |
| --- | --- |
| **Input** | **Output** |
| ivan prepend "hi"  pesho substring 0 4  pesho print  login pesho  pesho prepend "hi, again"  login pesho  pesho print  pesho prepend "HI!"  pesho length  pesho clear  pesho print  pesho undo  pesho print  end | 3  HI! |
| **Input** | **Output** |
| login pesho  login penka  login ivan  login mitko  login petyo  login petrov  login ivo  login stamat  login pencho  users  users pe  logout petrov  users pet  end | pesho  penka  ivan  mitko  petyo  petrov  ivo  stamat  pencho  pesho  penka  petyo  petrov  pencho  petyo |