Homework: Linear Data Structures – Lists

This document defines the **homework assignments** for the "Data Structures" course @ Software University. Please submit a single **zip** / **rar** / **7z** archive holding the solutions (source code) of all below described problems.

Problem 1. Sum and Average

Write a program that reads from the console a sequence of integer numbers (on a single line, separated by a space). Calculate and print the **sum** and **average** of the elements of the sequence. Keep the sequence in **List<int>**.

Input	Output
4 5 6	Sum=15; Average=5
1 1	Sum=1; Average=1
	Sum=0; Average=0
10	Sum=10; Average=10
2 2 1	Sum=5; Average=1.66666666666667

Problem 2. Sort Words

Write a program that reads from the console a **sequence of words** (strings on a single line, separated by a space). **Sort** them alphabetically. Keep the sequence in **List<string>**.

Input	Output
wow softuni alpha	alpha softuni wow
hi	hi
rakiya beer wine vodka whiskey	beer rakiya vodka whiskey wine

Problem 3. Longest Subsequence

Write a method that finds the **longest subsequence of equal numbers** in given **List<int>** and returns the result as new **List<int>**. If several sequences has the same longest length, return the leftmost of them. Write a program to test whether the method works correctly.

Input	Output
12 2 7 4 **3 3** 8	3 3
2 2 2 3 3 3	2 2 2
4 4 **5 5 5**	5 5 5
1 2 3	1
0	0

Problem 4. Remove Odd Occurences

Write a program that **removes** from given sequence all numbers that occur **odd number of times**.

Input	Output	Comments
1 **2 3 4** 1	1 1	2, 3 and 4 occur odd number of times (once). 1 occurs 2 times

1 2 3 **4 5** 3 **6** 7 **6** 7 **6**	3 3 7 7	1, 2, 4, 5 and 6 occurs odd number of times → removed
1 2 1 2 1 2		All numbers occur odd number of times → removed
3 7 **3 3** 4 **3** 4 **3** 7	7 4 4 7	3 occurs odd number of times (5) → removed
1 1	1 1	All numbers occur even number of times → sequence stays unchanged

Problem 5. Count of Occurences

Write a program that finds in given array of integers **how many times each of them occurs**. The input sequence holds numbers in range [0…1000]. The output should hold all numbers that occur at least once along with their number of occurrences.

Input	Output

`3 4 4 2 3 3 4 3 2`	`2 -> 2 times`
	`3 -> 4 times`
	`4 -> 3 times`

`1000`	`1000 -> 1 times`
`0 0 0`	`0 -> 3 times`

`7 6 5 5 6`	`5 -> 2 times`
	`6 -> 2 times`
	`7 -> 1 times`

Problem 6. Implement the Data Structure ReversedList<T>

Implement a data structure **`ReversedList<T>`** that holds a sequence of elements of generic type **T**. It should hold a **sequence of items in reversed order**. The structure should have some **capacity** that **grows twice** when it is filled. The reversed list should support the following operations:

- **`Add(T item)`** → adds an element to the sequence (grow twice the underlying array to extend its capacity in case the capacity is full)
- **`Count`** → returns the number of elements in the structure
- **`Capacity`** → returns the capacity of the underlying array holding the elements of the structure
- **`this[index]`** → the indexer should access the elements by **index** (in range **`0`** … **`Count-1`**) in the reverse order of adding
- **`Remove(index)`** → removes an element by **index** (in range **`0`** … **`Count-1`**) in the reverse order of adding
- **`IEnumerable<T>`** → implement an enumerator to allow iterating over the elements in a **`foreach`** loop in a reversed order of their addition

Hint: you can keep the elements in the order of their adding, by access them in reversed order (from end to start).

Problem 7. Implement a LinkedList<T>

Implement the data structure **singly linked list `LinkedList<T>`** that holds a sequence of linked elements. Define two classes:

- **`ListNode<T>`** holding the **value** and a pointer to the **next element**.
- **`LinkedList<T>`** holding the **first element** + operations **`Add(T item)`**, **`Remove(index)`**, **`Count`**, **`IEnumerable<T>`**, **`FirstIndexOf(T item)`**, **`LastIndexOf(T item)`**.

The **`LinkedList<T>`** is very similar to **`DoublyLinkedList<T>`** but holds a pointer to the next element only (not to both next and previous elements).

Problem 8. * Distance in Labyrinth

We are given a labyrinth of size N x N. Some of its cells are empty (**0**) and some are full (**x**). We can move from an empty cell to another empty cell if they share common wall. Given a starting position (*****) calculate and fill in the array the minimal distance from this position to any other cell in the array. Use "**u**" for all unreachable cells.
Example: