CS5004, Object-Oriented Design and Analysis

Lab 7: ADTs and Recursive Data Structures

Therapon Skoteiniotis, Tamara Bonaci and Abi Evans

Due Date: Refer to Canvas

1. Summary

In today's lab, we will continue our conversation about ADTs, and we will focus on recursive data structures. We will focus on:

- Recursive implementations of linked list
- Recursive implementation of sets

Note 1: Labs are intended to help you get started, and give you some practice while the course staff is present and able to provide assistance. You are not required to finish all the questions during the lab, but you are expected to push your lab work to a designated repo on the Khoury GitHub.

GitHub at the end of the lab.

Please check course website for Lab 7 deadline.

Problem 1: Recursive Implementation of a Linked List

Consider the following implementation of a List of Integers, available as a starter code for Lab 5. Using the available code, implement the following methods:

- Implement method `contains(Integer element)` on lists that returns `true` if element is in the list and `false` otherwise.
- Implement method `elementAt(Integer index)` that returns the element found at `index` in the list. Your code should deal with the situations where the value of `index` provided is outside the bounds of your list.

Problem 2: Recursive Implementation of a Set

Please provide the design and implement a data collection *Set*, as in the mathematical notion of a set. Here is the specification given to you describing all the operations on Set. The specification uses:

- {} to denote the empty set,
- {1,2,3} to denote the set with the elements 1, 2, 3.
- ∪ to denote union of two sets, i.e., a ∪ b.

Operation	Specification	Comments
`emptySet(): Set`	`emptySet() = {}`	• Creates an empty set.

`isEmpty() : Boolean`	`{}.isEmpty() = true`
	
	`{1,2,...}.isEmpty() = false`
		• Calling `isEmpty()` on an empty set must return `true`.
		• Calling `isEmpty()` on a non empty set must return `false`.

`add(Integer n) : Set`	`{}.add(n) = {n}`
	
	`aSet.add(n) = aSet,`
	`  if aSet.contains(n) = true`
	
	`aSet.add(n) = {n} ∪ aSet,`
	`  if aSet.contains(n) = false`
		• Calling `add(n)` on the empty set adds the new element `n` to the empty set.
		• alling `add(n)` on a non-empty set `aSet`returns the same set`aSet` if `n` is already a member of `aSet`.
		• Calling `add(n)` on a non-empty set `aSet`returns the union of `aSet` with the set `{n}`.

`contains(Integer n) : Boolean`	`{}.contains(n) = false`
	
	`aSet.contains(n) = true,`
	`  if n ∈ aSet`
		• Calling `contains(n)`on the empty set returns `false`.
		• Calling `contains(n)`on a non-empty set `aSet` returns `true` if `n` is a in the set `aSet`.

	aSet.contains(n) = false, if n ∉ aSet	• Calling contains(n) on a non-empty set aSet returns false if n is not in aSet
remove(Integer ele) : Set	{}.remove(x) = {} aSet.remove(x) = bSet, if aSet.contains(x) == true then aSet = bSet.add(x) aSet.remove(x) = aSet, if aSet.contains(x) == false	• Calling remove(x) on the empty set returns the empty set. • Calling remove(x) on a non-empty set aSetthat contains the element x returns the set bSet that has the same elements as aSetbut with the element xremoved. • Calling remove(x) on a non-empty set aSetthat does not contain the element x returns the set aSetunchanged.
size() : Integer	{}.size() = 0 aSet.size() = n, where |aSet| = n	• Calling size() on the empty set returns 0. • Calling size() on a non-empty set aSetreturns the number of elements in aSet

Optional Problem 3: List of List

Please design a Java program for a **list of list** of integers that supports the following operations.

1. **size** : returns the number of list of integers inside this list, e.g., ((), (1), (2 3)) returns 3
2. **length** : returns the number of total integers inside this list, e.g., ((), (), (2, 3)) returns 2
3. **sum** : returns the sum of all integers inside this list, e.g., ((), (1), (2 3)) returns 6
4. **isEmpty** : check if this list is empty, e.g., () returns true, (()) returns false
5. **add** : takes a list of integers and prepends (adds) it to this list of list of integers

6. **removeInteger** : takes an integer and removes the **first** occurrence of this integer in the list
7. **removeAllInteger** : takes an integer and removes the **all** occurrence of this integer in the list