Homework 8

You have to submit your solutions as announced in the lecture.
Unless mentioned otherwise, all problems are due 2017-04-06, 11:00.
There will be no deadline extensions unless mentioned otherwise in the lecture.

Problem 8.1 *List Sets*

Points: 8

Implement an abstract class for iterable sets. The iterator should return the elements in the order of insertion.
Extend it to a concrete class for mutable list-backed sets.
Write a test program that

- creates a set of integers,
- inserts some elements,
- prints all elements using the iterator.

Depending on your programming language, this might look as follows:

```
abstract class Set[A]() extends Iterable[A]
    fun contains(x : A) : bool
    fun insert(x : A) : unit
    fun delete(x : A) : unit
    fun iterator() : Iterator[A]

class ListSet[A]() extends Set[A]
    private elements := Nil[A]                    the immutable linked list backing the set, initially empty

    fun contains(x : A) : bool =
        ...
    fun insert(x : A) : unit =
        ...
    fun delete(x : A) : unit =
        ...
    fun iterator() : Iterator[A] =
        ...

test := new ListSet[int]()
test.insert(4)
test.insert(2)
test.insert(4)
foreach(test.iterator, x ↦ print x)                prints 2,4 or 4,2
```

Problem 8.2 *Binary Search Trees*

Points: 8

Implement an iterable data structure for $BST[int, \leq]$. The iterator should return the elements in $\leq$-order.
Write a test program that

- creates a set of integers,
- inserts some elements,
- prints all elements using the iterator.

If you combine it with the previous problem, this might look as follows:

```
class IntBST() extends Set[int]
                                        the tree backing the set, initially a single node
    private elements : Tree[Option[int]] = new Tree[int](None, Nil)
```

```
fun contains(x : A) : bool =
    ...
fun insert(x : A) : unit =
    ...
fun delete(x : A) : unit =
    ...
fun iterator() : Iterator[A] =
    ...
```

Make sure that

- the implementations of *insert* and *delete* preserve the necessary BST property,
- the implementations of *insert*, *delete*, and *contains* are logarithmic in the best case.

Problem 8.3 *Hash Sets*

Points: 8

Implement an abstract class for $HashSet[A]$ sets for an arbitrary hash function $hash : A \to \mathbb{Z}_m$. The iterator may return elements in any order. Extend it to a concrete class $LastDigitHashSet$ that hashes integers based on their last digit.

Write a test program that

- creates a $LastDigitHashSet$ of integers,
- inserts some elements,
- prints all elements using the iterator.

Depending on your programming language, this may look as follows:

```
abstract class HashSet[A](m : int) extends Set[A]
                                        the array backing the hash set, initially all buckets empty
    private bucket : Array[ListSet[A]] = new Array[ListSet[A]](m)
    for x from 0 to m − 1
        bucket[i] := new ListSet[A]()

                                        the abstract hash function that subclasses must provide
                                        postcondition: return value between 0 and m − 1
    fun hash(x : A) : int

                                        the functions implemented in this class
    fun contains(x : A) : bool =
        ...
    fun insert(x : A) : unit =
        ...
    fun delete(x : A) : unit =
        ...
    fun iterator() : Iterator[A] =
        ...

class LastDigitHashSet() extends HashSet[int](10)
    fun hash(x : A) : int = {x mod 10}
```

Make sure that your implementation exploits the hashing, i.e., *contains*(x), *insert*(x), and *delete*(x) should only access *bucket*$[hash(x)]$.