IMPERATIVE PROGRAMMING HT2018

SHEET 4

GABRIEL MOISE

Question 1

```
object Question1
{
 def main(args: Array[String]) =
 {
  val myset = new scala.collection.mutable.HashSet[String]
  // myset: scala.collection.mutable.HashSet[String] = Set()
  assert (myset.size == 0)
  assert (myset.isEmpty == true)
  // In the beginning, the size of myset was 0, as expected and therefore myself is empty
  assert (myset.contains("a") == false)
  myset.add("a") ; myset.add("ab")
  assert (myset.size == 2)
  assert (myset.isEmpty == false)
  // scala.collection.mutable.HashSet[String] = Set(ab, a)
  assert (myset.contains("a") == true)
  assert (myset.contains("b") == false)
  // So, the contains function behaves as we expect
  assert (myset.add("a") == false)
  assert (myset.size == 2)
  // The first assertion is false because we cannot add "a", as it already is in the set
  assert (myset.add("b") == true)
  assert (myset.size == 3)
  // The first assertion is true because "b" is not in myset, so the function also adds "b" to myset
  // scala.collection.mutable.HashSet[String] = Set(ab, a, b)
  assert (myset.remove("c") == false)
  assert (myset.isEmpty == false)
  // This is false as we cannot remove a value which is not in myset
  assert (myset.remove("ab") == true)
  assert (myset.size == 2)
  // This is true and "ab" is removed from myset
  // scala.collection.mutable.HashSet[String] = Set(a, b)
  myset.remove("a") ; myset.remove("b")
  // scala.collection.mutable.HashSet[String] = Set()
```

```
    assert (myset.size == 0)
    assert (myset.isEmpty == true)
    // Now, a function that throws an exception is max, which returns the biggest element from the set (here in lexicografic order):
    // myset.add("a") ; myset.add("b") ; myset.add("ab") ; myset.add("ba")
    // scala> myset
    // res17: scala.collection.mutable.HashSet[String] = Set(ab, ba, a, b)
    // scala> myset.max
    // res18: String = ba
    // But when the set is empty, it throws and exception:
    // myset.remove("a") ; myset.remove("b") ; myset.remove("ab") ; myset.remove("ba")
    // scala> myset
    // res23: scala.collection.mutable.HashSet[String] = Set()
    // myset.max
    // java.lang.UnsupportedOperationException: empty.max
    // because the precondition for max is not respected, more precisely the precondition is myset.size > 0.
  }
}
```

Question 2

```
// The interface of a stack, which represents a sequence of data (in our case, of type A)
trait Stack[A]
{
  // state = stack -> a sequence of data (elements) of type A that are added in the front and removed from the front
  // init = stack = {}

  // Add an element of type A to the stack
  // Post : stack = elem : stack_0 (Haskell notation)
  def push (elem : A)

  // Remove the most-recently added element from stack and return it
  // Pre : stack is not empty, otherwise we throw an exception
  // Post : stack = tail (stack_0) ∧ return head (stack_0) (Haskell notation)
  def pop : A

  // Check if stack is empty or not
  // Post : stack = stack_0 ∧ return true if the stack is empty and false otherwise (we can count how many elements we currently have and simply test count == 0)
  def isEmpty : Boolean
}
```

Question 3

```
/** state: S : P [0..N)
  * init: S = {} */
trait IntSet
{
  /** Add elem to the set if 0 <= elem < N.
    * pre: S0 ⊆ [0..N)
    * (a) post: S = S0 ∪ {elem} && S ⊆ [0..N) */
  def add(elem: Int)

  /** Does the set contain elem?
    * post: S = S0 ∧ returns elem ∈ S */
  def isIn(elem: Int): Boolean

  /** Remove elem from the set.
    * post: S = S0 −{elem} */
  def remove(elem: Int)

  /** The size of the set.
    * post: S = S0 ∧ returns #S */
  def size : Int
}
////////////////////////////////////////////////////////////////////////////////
class BitMapSet
{
  // Abs : set = {x ∈ [0..N) | a(x) = true}
  // DTI : a(i) = true <=> i ∈ set for all 0 <= i < N && count = # {a(i) = true | 0 <= i < N}

  // The number of elements currently in the stack
  private var count = 0
  // The biggest element that can appear in the set
  private val N = 10000
  // The Boolean-type array of size N
  private val a = new Array [Boolean] (N)
  // Setting every element to false as initially the set is empty
  for (i <- 0 until N) a(i) = false

  // Adds elem to the set by setting a(elem) to true (if elem was already in the set nothing changes)
  // Pre : 0 <= elem < N
```

```
 // Post : a(i) = a0(i) for all 0 <= i < N with i/=elem, and a(elem) = true and count += 1 if a0(elem) = false
 def add (elem : Int) =
  {
   require (0 <= elem && elem < N)
   if (!a(elem)) count += 1
   a(elem) = true
  }

 // The value of a(elem) tells whether elem is in the set or not
 // Pre : 0 <= elem < N
 // Post : a = a0 (we don't modify the array in any way)
 def isIn (elem : Int) : Boolean = a(elem)

 // Removes elem from the set by setting a(elem) to false (if elem was not in the set nothing happens)
 // Pre : 0 <= elem < N
 // Post : a(i) = a0(i) for all 0 <= i < N with i/=elem, and a(elem) = false and if a0(elem) = true, count -= 1
 def remove (elem : Int) =
  {
   require (0 <= elem && elem < N)
   if (a(elem)) count -= 1
   a(elem) = false
  }

 // Counts the values of true among the boolean-typed array a
 // Post : count
 def size : Int = count
}
```

Question 4

```
/**
```

(a) First of all, by definition, a mathematical set is an UNORDERED sequence of elements, therefore the API description for the Scala Set[A] trait is ambiguous because we can't define the "first" element of a set without having an order. By doing some experiments I found out that it behaves in a non-intuitive way, the implementation method being not obvious at all. For example, I did:

scala> val set = scala.collection.mutable.Set[Int]()

scala> set.add(1) ; set.add(2) ; set.add(-4)

res3: scala.collection.mutable.Set[Int] = Set(1, 2, -4)

scala> set.head

res4: Int = 1 (so the head is not the smallest element)

Then I added some other numbers and it got even weirder:

scala> set.add(-1) ; set.add(-20) ; set.add(-30) ; set.add(-50)

```
res12: scala.collection.mutable.Set[Int] = Set(-50, 1, -20, 2, -1, -30, -4)

So it doesn't seem like there is an obvious order. But when I add 0, every time I do that, it becomes the head and it stays that way:

scala> set.add(0)

res14: scala.collection.mutable.Set[Int] = Set(0, -50, 1, -20, 2, -1, -30, -4)

scala> set.add(12) ; set.add(10) ; set.add(8)

res18: scala.collection.mutable.Set[Int] = Set(0, 12, -50, 1, -20, 2, -1, -30, 10, -4, 8)

scala> set.head

res19: Int = 0

*/

/**

(b) So, in my opinion, it would be sensible, since we only have elements in the set that are in [0..N), as in Question 3, to define the first element of the set as the smallest one, like the set is an ordered sequence, the problem appears when the set is empty, then an assertion is thrown in "Scala java.util.NoSuchElementException: next on empty iterator": */


// Calculating the head of the set (which we considered ordered by the "bit map" we created in Question 3)

// Pre : set is non-empty, otherwise we throw an exception

// Post : the smalles element of the set, as min{ i | 0 <= i < N such that a(i) = true}

def head : Int


// (c)


def head : Int =

 {

   var i = 0

   // Invariant I : a[0..i) = false && 0 <= i <= N

   // Variant N-i

   while (i < N && a(i) == false) i += 1

   assert (i < N) // throw an exception if the set is empty

   i // otherwise return the first i such that a(i) = true

 }
```

Question 5

```
/** Delete the number stored against name (if it exists)

     * (a) post : book = book0 - {name -> number} (if (name ∈ dombook)) || book = book0 (if  otherwise) && returns (name ∈ dombook) */

   def delete(name: String) : Boolean

////////////////////////////////////////////////////////////////////////////////

/** Delete the number stored against name (if it exists) and return whether the name was in the phone book or not */

  def delete(name: String) : Boolean =

   {

     val pos = find (name) // finding the position of name in names
```

```
    if (i!=count)
    {
      count -= 1
      names(i) = names(count)
      numbers(i) = numbers(count)
      true
    }
    else false
  }
```

Question 6

```
// Representing the phone book using a pair of arrays


object ArraysBook extends Book{
  private val MAX = 1000 // max number of names we can store
  private val names = new Array[String](MAX) // which will be ordered lexicographically
  private val numbers = new Array[String](MAX) // the corresponding array
  private var count = 0
  // Abs : These variables together represent the mapping { names(i) -> numbers(i) | i <- [0..count) }
  // DTI : count <= MAX && entries in names[0..count) are in a strictly-increasing order


  // pre: names is sorted
  // post: the index i < count such that names(i) = name or return count if no such index exists
  // Time Complexity: O(log2(count))
  private def find(name: String) : Int = {
    var left = 0 ; var right = count
    // Invariant: names[0..left) < name <= names[right..count)
    while (left < right)
    {
      val middle = (left + right) / 2 // left <= middle < right
      if (names(middle) < name) left = middle + 1
        else right = middle
    }
    // left == right => names[0..left) < name <= [left..count), but we have to check if the name appears in the array, or not
    // Therefore, we must check that names(left) = name:
    if (name == names(left)) left
      else count // if not, as we said, we return count
  }


  // Time complexity: O(log2(count))
```

```
  /** Return the number stored against name */
  def recall(name: String) : String = {
    val i = find(name)
    assert(i<count)
    numbers(i)
  }

  // Time complexity: O(log2(count))
  /** Is name in the book? */
  def isInBook(name: String) : Boolean = find(name)<count

  // Here, we will implement a binary search, and there are 2 possibilities:
  // 1. name is an element of names and we therefore update the number with the new one
  // 2. name is not an element of names, but we find left such that names[0..left) < name <= names[left..count), therefore we will insert name there with the corresponding number
  // Time complexity: O(count) for the insertion part (the search is o(log2(count))
  def store(name: String, number: String) = {
    var left = 0 ; var right = count
    // Invariant: names[0..left) < name <= names[right..count)
    while (left < right)
    {
      val middle = (left + right) / 2 // left <= middle < right
      if (names(middle) < name) left = middle + 1
        else right = middle
    }
    // left == right => names[0..left) < name <= [left..count) and we do what we described above:
    if (name == names(left)) numbers(left) = number
    // Otherwise, to insert an element in the position left, we have to shift all the elements from left,left+1...count-1 by one and increase count by 1
      else if (count<MAX)
      {
        var i = count
        // names[left+1..count+1) = names0[left..count) && numbers[left+1..count+1) = numbers0[left..count)
        while (i>left) {names(i) = names(i-1) ; numbers(i)=numbers(i-1) ; i -= 1}
        names(left) = name ; numbers(left) = number
        count += 1
      }
  }

  // We first search if name is in names and then we delete it by shifting the elements from the right by one position to the left and decrease count by 1, otherwise we simply return false
```

```
  // Time complexity: O(count) for the shifting part
  def delete (name : String) : Boolean ={
    var pos = find(name)
    if (pos<count)
    {
      var i = pos
      // names[pos..count-1) = names0[pos+1..count) && numbers[pos..count-1) = numbers0[pos+1..count)
      while (i<count-1) {names(i) = names(i+1) ; numbers(i) = numbers(i+1) ; i += 1}
      count -= 1
      true
    }
    else false
  }
}
```

Question 7

```
// The interface of the bag
// Each implementation of this trait represents a mapping from integers to the number of times they appear in the bag

trait Bag {
  // state : bag : Int -> Int
  // init : bag = {0->0 ,1->0, 2->0, 3->0... (MAX-1)->0}, or bag(x) = 0 for all 0 <= x < MAX

  // Add the element x to the bag
  // Pre : 0 <= x < MAX
  // Post : bag(x) increases by 1 and bag(i) remains the same for all 0 <= i < MAX with i /= x
  def add (x : Int)

  // Find the number of copies of x in the bag
  // Pre : 0 <= x < MAX
  // Post : the bag array remains the same and we return bag(x)
  def copies (x : Int) : Int
}
////////////////////////////////////////////////////////////////////////////////////////
// The concrete state represents the mapping bag = {bag(0), bag(1) ...bag(MAX-1)}
// Abstract function : bag = {i -> bag(i) | 0 <= i < MAX}
// DTI : bag(i) = the number of times i appears in the bag for all 0 <= i < MAX

object ArrayBag extends Bag{
  private val MAX = 10000 // the maximum value of an integer that is put in the bag (actually that is MAX-1)
```

```
  private val c = new Array [Int] (MAX)

  // As the initial state suggests, the c array contains only 0 in the beginning:
  for (i <- 0 until MAX) c(i) = 0

  // Add the element x to the bag
  // Pre : 0 <= x < MAX
  // Post : c(x) increases by 1 and c(i) remains the same for all 0 <= i < MAX with i /= x
  def add (x : Int) = c(x) += 1

  // Find the number of copies of x in the bag
  // Pre : 0 <= x < MAX
  // Post : the c array remains the same and we return c(x)
  def copies (x : Int) : Int = c(x)
}
```

Question 8

```
// Main module for the array bag example, using an implementation of Bag
import scala.io.StdIn.readLine

object SortingBag
{
  def main (args: Array[String]) =
    {
      val MAX = 10000
      // The array bag which we will play with, by adding elements to it and demanding the sorted array with what we have added so far or the number of copies of a given i
      val bag = ArrayBag
      var done = false // are we finished yet?
      while(!done)
        {
          val cmd = readLine("Add (a), copies (c), sort (s) or quit (q)? ")
          cmd match{
        case "a" => {
          val elem = (readLine("Element to add to the bag? ")).toInt
          if (elem < MAX) bag.add(elem)
          else println(elem+" exceeds the maximum value allowed.")
        }
        case "c" => {
          val elem = (readLine("What element to count? ")).toInt
```

```
        println("The element "+elem+" appears "+bag.copies(elem)+" times in the bag.")
      }
      // Here, as the question asks, we will do this in O(N+MAX)
      case "s" => {
        println("The bag contains: ")
        for (i <- 0 until MAX)
          for (j <- 0 until bag.copies(i)) print(i+" ")
        println()
        // The time complexity for this is O(N+MAX) as we go through the array from 0 until MAX and as we have N elements in the bag, the sum of all the elements in the array will be equal to N, so we will print N elements in total. (MAX comes from the for loop because i increases by 1 MAX-1 times and we have N operations of print)
      }
      case "q" => done = true
      case _ => println("Please type `a', `c', `s' or `q'.")
    }
  }
 }
}
/**
Add (a), copies (c), sort (s) or quit (q)? a
Element to add to the bag? 1
Add (a), copies (c), sort (s) or quit (q)? a
Element to add to the bag? 3
Add (a), copies (c), sort (s) or quit (q)? a
Element to add to the bag? 1
Add (a), copies (c), sort (s) or quit (q)? a
Element to add to the bag? 4
Add (a), copies (c), sort (s) or quit (q)? c
What element to count? 1
The element 1 appears 2 times in the bag.
Add (a), copies (c), sort (s) or quit (q)? c
What element to count? 5
The element 5 appears 0 times in the bag.
Add (a), copies (c), sort (s) or quit (q)? s
The bag contains:
1 1 3 4
Add (a), copies (c), sort (s) or quit (q)? a
Element to add to the bag? 20
Add (a), copies (c), sort (s) or quit (q)? a
Element to add to the bag? 20
```

Add (a), copies (c), sort (s) or quit (q)? a

Element to add to the bag? 15

Add (a), copies (c), sort (s) or quit (q)? s

The bag contains:

1 1 3 4 15 20 20

*/