IMPERATIVE PROGRAMMING 2015

MOISE GABRIEL

QUESTION 1

(a)

```
> data Tree = Leaf | Fork Tree Tree

> preorder :: Tree -> [Bool]
> preorder Leaf = [False]
> preorder (Fork l r) = True : (preorder l) ++ (preorder r)
```

(b)

```
abstract class Tree
case class Leaf() extends Tree
case class Fork ( le : Tree , ri : Tree) extends Tree

def preorder (node : Tree , stepi : int , dest: Array [Boolean]) : int = {
    node match {
        case Leaf () => { dest(stepi) = false ; stepi + 1 }
        case Fork (le : Tree, ri : Tree) => {
            var step = stepi
            dest(step) = true ; step += 1
            step = preorder ( le, step, dest)
            step = preorder (ri, step , dest)
            step
        }
    }
}
```

(c)

```
def iter_preorder (node : Tree, dest : Array [Boolean], stk : Array [Tree]) : int = {
    var current = node
    var i = 0
    var stk_len = 0
    var ok = true
    while (ok == true) {
      current match {
          case Leaf () => {
              dest(i) = false
              if (stk_len > 0)
                { stk_len -= 1 ; current = stk(stk_len)}
```

```
                    else    ok = false
                }
                case Fork (le : Tree, ri : Tree) => {
                    dest (i) = true
                    stk (stk_len) = ri
                    stk_len += 1
                    current = le
                }
            }
            i += 1
        }
        :
    }
```

(e)

```
def toTree (preord : Array [Boolean], stepi : int) : (Tree, int) = {
    var step = stepi
    if (preord(step) == false) { (Leaf (), stepi + 1 }
    else {
        val left_son = toTree (preord , stepi + 1)
        val right_son = toTree (preord, left_son._2)
        (Fork (left_son._1, right_son._1), right_son._2)
    }
}
```