QUESTION 3

```
class fraction {
  private var list = new fraction.Node (0, null) // Dummy header
  private var end = list

  // Function that allows the user to add terms to the "continued fraction"
  def add (x: Int): Unit = {
      var n1 = new fraction.Node (x, null)
      end.next = n1
      end = n1
  }

  // (a)
  def print: String = {
      var current = list.next
      var par = 0 // we count how many brackets we open to see how many we need to close
      var str = ""                                                    at the end
      while (current != null)
      { if (current.next == null) str = str + current.datum // the last term of the sequence
        else if (current.next.next == null) str = str + current.datum + "+1/"
        else {str = str + current.datum + "+1/("; par += 1}
        current = current.next
      }
      for (i <- 0 until par) str = str + ")"
      str
  }
  // (b)
  def nat2cfRec (n: Int, d: Int): String = {
      if (n%d == 0) return (n/d).toString // exact division
      else if (n%d == 1) return (n/d).toString + "+1/" + nat2cfRec (d, n%d)
      else return (n/d).toString + "+1/(" + nat2cfRec (d, n%d) + ")"
  }
```

```
// (c)
def nat2cf (n1: int, d1: int) : String = {
    var frac = new fraction
    var n = n1
    var d = d1
    while (d != 0)
    { var a = n / d
      var b = n % d
      frac.add (a)
      n = d
      d = b
    }
    frac.print
}
// (d)
// This function deletes the first term of the continued fraction, so that we keep recursing on the list
def del : Unit = list.next = list.next.next
def cf2natRec : (int, int) = {
    var x = list.next.datum
    if (list.next.next = null) return (x, 1)
    else
    { this.del
      var (n1, d1) = this.cf2natRec
      return (n1 * x + d1, n1)
    }
}
// (e)
def cf2nat : (int, int) = {
    var current = list.next
    var (m1, m2, m3, m4) = (current.datum, 1, 1, 0)
    var n = 0 ; var d = 0
    if (current.next == null) return (current.datum, 1) // the result is an integer
    else current = current.next
```

```
    while (current != null)
    {
      if (current.next == null) { n = current.datum ; d = 1 }
      else //the matrix multiplication (m1 m2)(current.datum 1)
                                       (m3 m4)(1          0)
      {
        var p1 = m1 * current.datum + m2
        var p2 = m1
        var p3 = m3 * current.datum + m4
        var p4 = m3
        m1 = p1 ; m2 = p2 ; m3 = p3 ; m4 = p4
      }
      current = current.next
    }
    return (m1*n + m2*d , m3*n + m4*d)  // (m1 m2)(n)
                                        (m3 m4)(d)
  }
}

// Companion object
object fraction {
    private class Node (var datum: Int, var next: Node)
}
```