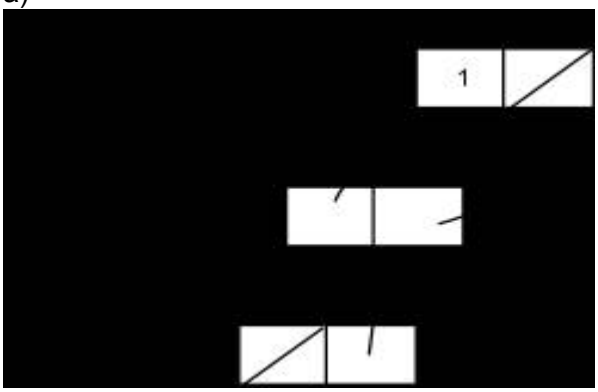
Soluciones Parcial 3 LPP 2011-2012

Ejercicio 2

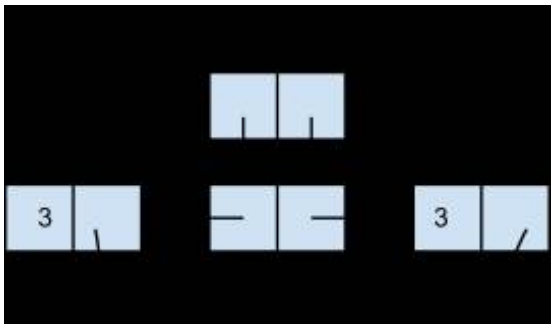
a)



b)

```
(define p1 (cons 3 '())
(define p2 (cons '() p1))
(define p3 (cons p1 p2))
(define x (cons p3 p3))
(set-cdr! p1 x)
(set-car! p2 x)
```

c)



d) Hay varias soluciones. Por ejemplo:

```
(set-cdr! (caar x) (car x))
(set-cdr! (cddr x) (car x))
```

Ejercicio 3

```
(define (convertir! exp-s)
   (cond
      ((null-exp-s? exp-s) #t)
      ((null-exp-s? (rest-exp-s exp-s)) #t)
      ((leaf-exp-s? (first-exp-s (rest-exp-s exp-s)))  (convertir! (rest-exp-s exp-s)
))
      (else  (let ((sig (rest-exp-s (rest-exp-s exp-s))))
                  (begin
                    (set-cdr! (rest-exp-s exp-s) (first-exp-s (rest-exp-s exp-s)))
                    (set-car! (rest-exp-s exp-s) '<)
                    (set-cdr! (ult-pareja (rest-exp-s (rest-exp-s exp-s))) (cons '>
sig))
                    (convertir! exp-s))))))

(define (ult-pareja exp-s)
  (if (null-exp-s? (rest-exp-s exp-s)) exp-s
      (ult-pareja (rest-exp-s exp-s))))
```

Ejercicio 4

a)
```
def makeContador(x:Int, f:(Int)=>Int):()=>Int = {
         var cuenta = x
         ()=>{
           cuenta = f(cuenta)
           cuenta
         }
 }
```

b)

Ejercicio 5

a) La definición incorrecta es la b), porque Clase3 no es subclase de Clase1. El Trait2 debe mezclarse con una clase que sea de tipo Clase1, por tanto no es posible mezclar la Clase3 con el Trait2.

b)
a.h(2,4) → 8
a.g(2,4) → 106
~~b.g(2,4)~~
~~b.h(2,4)~~
c.g(2,4) → 116
c.h(2,4) → 8
d.f(2) → Error: f no es un método de Clase1

Ejercicio 6

a) La sentencia

```
val pong = new Pong
```

crea un nuevo agente de la clase *Pong* y lo guarda en la variable *pong*. Después creamos otro agente de la clase *Ping* al que se le pasa como parámetro de inicialización el agente *pong* y el valor 10. La llamada al método *start* en ambos agentes crea dos hilos concurrentes. El primer hilo (el objeto *ping*) hace un bucle en el que envía 10 mensajes (cadena "Ping") al segundo. El

envío de mensajes no bloquea, por lo que los 10 mensajes quedan almacenados en el buzón de *pong* a la espera de que los procese. El segundo hilo (el objeto *pong*) realiza un *receive* esperando un mensaje. La llamada a *receive* bloquea y sólo se sale de ella cuando se recibe el mensaje. El mensaje se consume y al no hacer más llamadas a *recieve* quedan en el buzón 9 mensajes sin consumir (más el mensaje "stop")

En la salida estándar se escribe:

```
Ping envía Ping (10 veces)
Recibido mensaje
```

b)

```
import scala.actors.Actor._

def pong() = {
   actor {
      receive {
         case "Ping" =>
           println("Recibido mensaje")
         case "Stop" =>
      }
   }
}

def ping(count:Int, pong: scala.actors.Actor) = {
   actor {
      for(i <- 1 to count) {
         println("Ping envía Ping")
         pong ! "Ping"
      }
      pong ! "Stop"
   }
}

ping(10,pong())
```