Nombre:

Lenguajes y Paradigmas de Programación
Curso 2010-2011
Segundo parcial, turno de tarde

Normas importantes

- La puntuación total del examen es de 20 puntos.
- Se debe contestar cada pregunta en las hojas que entregamos. Utiliza las últimas hojas para hacer pruebas. No olvides poner el nombre.
- La duración del examen es de 1 hora.

Ejercicio 1 (5 puntos)

1) (3 puntos) Supongamos las siguiente definiciones:

```
class Vehiculo
class Camion extends Vehiculo
class Furgoneta extends Camion
class Coche extends Vehiculo
def transportar(c:Camion, lista:List[Coche]): Camion = {...}
var lista1 = List(new Vehiculo, new Coche)
var lista2 = List(new Coche, new Coche)
var peugeot = new Vehiculo
var scania: Camion = new Furgoneta
var honda = new Camion
```

Indica qué sentencias son erróneas en Scala y explica por qué lo son.

```
a) var p = transportar(scania, lista1)
b) var p = transportar(honda, lista2)
c) var p2: Vehiculo = transportar(scania, lista2)
d) var p = transportar(peugeot, lista2)
```

2) (2 puntos) Rellena los huecos para obtener una expresión válida en Scala y pon un ejemplo para ilustrar su funcionamiento:

```
def funcion1(a: Int, g: __________): ___________ = {
  _________ => “hola” + g(a,a)
}
```

Ejercicio 2 (5 puntos)

Define en Scala la función recursiva pura (sin recursión por la cola) `aplica3D` que reciba una lista de coordenadas 3D en forma de tupla, una función unaria y una coordenada (“x”, “y” ó “z”). Deberá aplicar la función a los elementos correspondientes a esa coordenada y devolver las nuevas coordenadas.

Ejemplo:

```
val lista = List((1,2,3), (4,5,6), (7,8,9), (10,11,12))
def suma(x: Int) = x + 2
aplica3D(lista, suma _, “y”) → ((1,4,3), (4,7,6), (7,10,9), (10,13,12))
```

Ejercicio 3 (5 puntos)
Dibuja y explica el diagrama de entornos de las siguientes instrucciones. ¿Qué devuelve la última expresión?

```
var x = 1
var y = 2

def f() = {
 var x = 10
 (y: Int) => {
     x = x + y
     x
  }
}

def g(h:Int=>Int) = {
 var x = 20
 h(100)
}

g(f())
```

Ejercicio 4 (5 puntos)

Implementa en Scheme el procedimiento `list->exprs!` que reciba una lista de un nivel y la mute, sin crear nuevas parejas, de forma que aparezcan nuevos niveles en aquellos lugares encerrados entre los símbolos < y >. Suponemos que la lista pasada como argumento siempre será correcta y la profundidad de los subniveles como máximo será 1. Explica, utilizando diagramas de caja y puntero, el funcionamiento de tu solución.

Ejemplo:

```
(define lista '(1 2 < 3 4 > 5 6 < 7 8 9 >))
(list->exprs! lista)
lista → (1 2 (3 4) 5 6 (7 8 9))
```