Lenguajes y Paradigmas de Programación

Curso 2010-2011
Examen Final de la convocatoria de junio

Normas importantes

- La puntuación total del examen es de 50 puntos sobre los 100 que se valora la nota de la asignatura.
- Se debe contestar **cada pregunta en una hoja distinta**. No olvides poner el nombre en todas las hojas.
- La duración del examen es de 3 horas.
- Las notas estarán disponibles en la web de la asignatura el viernes 17 de junio.
- La revisión será el lunes 20 de junio a las 10:00h en el despacho de Domingo Gallardo.

Pregunta 1 (8 puntos)

a) (3 puntos) Describe las características principales de los siguientes paradigmas de programación, e indica dos lenguajes de programación que se puedan ubicar en cada uno de los paradigmas:

- Paradigma funcional
- Paradigma declarativo
- Paradigma imperativo
- Paradigma orientado a objetos

b) (3 puntos) Define y escribe un ejemplo en Scala de los siguientes conceptos:

- Coercion
- Sobrecarga
- Polimorfismo en tiempo de ejecución

c) (2 puntos) Dado el siguiente ejemplo:

```
abstract class Bar { def bar(x: Int) : Int }
class Foo extends Bar { def bar(x: Int) = x }

trait Foo1 extends Foo { abstract override def bar(x: Int) = super.bar(x * 3) }
trait Foo2 extends Foo { abstract override def bar(x: Int) = super.bar(x + 3) }
trait Foo3 extends Foo { abstract override def bar(x: Int) = x + super.bar(x) }
trait Foo4 extends Foo { abstract override def bar(x: Int) = x * super.bar(x) }
```

Rellena los huecos con el resultado de las siguientes llamadas y explica tu respuesta:

```
scala> (new Foo with Foo2 with Foo4).bar(5)
scala> __________

scala> (new Foo with Foo2 with Foo1).bar(5)
scala> __________

scala> (new Foo with Foo3 with Foo4).bar(5)
scala> __________
```

Pregunta 2 (7 puntos)

Supongamos que estamos desarrollando en Scheme un programa de estadística y que guardamos una lista de frecuencias (número de veces que se repite un valor) como una lista de parejas (`<valor> <veces>`). Por ejemplo, la lista `((1.3) (2.4) (4.8))` representa que el valor 1 ha aparecido 3 veces, el 2 4 veces y el 4 8 veces.

La media de estos valores se puede expresar como

$(v_1*n_1 + v_2*n_2 + \ldots v_n*n_n)/(n_1+n_2+...+n_n)$,

siendo v_i el valor i-ésimo y n_i su frecuencia. Por ejemplo, en el caso anterior, la media de los valores es:

`(1*3+2*4+4*8)/3+4+8 = 2,87`

Define una función recursiva (`frecuencias lista-frecs`) que devuelva una pareja donde la parte izquierda corresponda a $(v_1*n_1 + v_2*n_2 + \ldots v_n*n_n)$ y la parte derecha a $(n_1+n_2+...+n_n)$. Utiliza esta función para calcular la media. Ejemplo:

```
> lista-frec
((1 . 3) (2 . 4) (4 . 8))
> (frecuencias lista-frec)
(43.15)
> (media lista-frec)
2,87
```

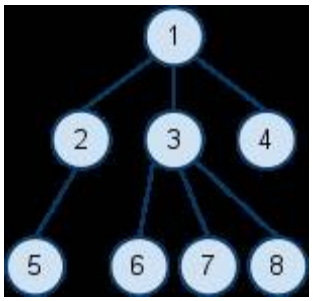
Pregunta 3 (9 puntos)

a) (3 puntos) Define en Scheme la barrera de abstracción del árbol genérico y utilízala para implementar el procedimiento (`equal-tree? tree1 tree2`) que reciba dos árboles genéricos como argumento y devuelva `#t` si son iguales y `#f` en caso contrario.

b) (6 puntos) Vamos a implementar en Scala el tipo `Tree` (como un árbol genérico). Su implementación es:

```
class Tree (val dato: Int, val hijos: List[Tree])
```

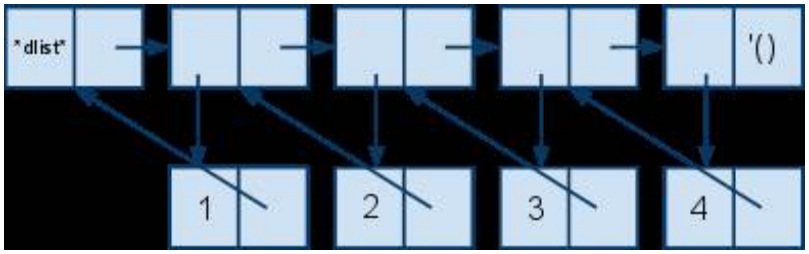
b.1) Define el siguiente árbol utilizando el tipo de dato `Tree`:



b.2) Añade al tipo `Tree` el método `sumaTotalTree` que reciba un `Tree` como argumento y devuelva la suma de todos sus nodos.

Pregunta 4 (9 puntos)

Vamos a construir un nuevo tipo de dato, llamado lista doblemente enlazada o `dlist`. Esta estructura de datos tiene la propiedad de que, en cada momento, se puede acceder tanto al siguiente elemento como al anterior. Para implementarlo, definimos los elementos de esta lista como parejas cuya parte izquierda contiene el dato propiamente dicho y la parte derecha el enlace al elemento anterior. La lista va precedida de una pareja que hace el papel de cabecera y que es referenciada por las variables que apuntan a ella. En la figura puedes ver esta implementación:



a) (6 puntos) Define el procedimiento `(make-dlist)` que construya una lista vacía y el procedimiento mutador `(add-dlist! dlist x)` que añada un elemento `x` en la primera posición de una lista creada por `make-dlist`.

b) (3 puntos) Define el procedimiento mutador `(insert-dlist! lista pos-dlist x)` que inserte un elemento x en la posición de la lista indicada por `pos-dlist`. Suponemos que la posición está contenida en la lista.

Por ejemplo, el siguiente código genera la figura anterior:

```
(define lista (make-dlist))
(add-dlist! lista 4)
(add-dlist! lista 3)
(add-dlist! lista 1)
(insert-dlist! lista 1 2)
```

Pregunta 5 (9 puntos)

Estudia el siguiente código en Scala:

```
def foo(lista: List[(Int) => Boolean]) = {
   (x: Int) => {
      var todos = true
      for (f <- lista)
         if (!lista(f)(x)) todos = false
      todos
   }
}
```

a) (6 puntos) Explica qué hace la función: qué recibe y qué devuelve, y escribe un ejemplo de código en Scala en el que se utilice la función

b) (3 puntos) Tiene un error de compilación. Encuéntralo y corrígelo.

Pregunta 6 (8 puntos)

Supongamos el siguiente código en Scala:

```
def constructor() = {
   var x = 0
   def suma(y:Int) = {
      x = x+y
      x
   }
   def resta(y:Int) = {
      x = x-y
      x
   }
   def getFunc(s:String) = {
      s match {
         case "suma" => suma _
         case "resta" => resta _
      }
   }
   getFunc _
}
____________
____________
____________
f(20) -> 20
g(10) -> 10
```

a) (4 puntos) Rellena los huecos (una sentencia en cada uno) para que las dos últimas expresiones devuelvan los valores indicados.
b) (4 puntos) Dibuja y explica el diagrama de entornos creado.