

# Lenguajes y Paradigmas de Programación

## Curso 2004-2005

### Examen de la Convocatoria de Junio

#### Normas importantes

- La puntuación total del examen es de 50 puntos que sumados a los 10 puntos de las prácticas dan el total de 60 puntos sobre los que se valora la nota de la asignatura.
- Para sumar los puntos de las prácticas **es necesario obtener un mínimo de 20 puntos en este examen.**
- Se debe contestar cada pregunta **en un 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 próximo día 1 de Julio.

#### Pregunta 1 (8 puntos)

a) (5 puntos)

Supongamos el siguiente procedimiento:

```
(define (h n)
  (lambda (x y)
    (set! x (+ (car n) (cdr n) y))
    (set-car! n x)
    (set-cdr! n y)
    x))
```

Escribe un ejemplo completo (en el que se llegue a evaluar el cuerpo del lambda) de uso del procedimiento, incluyendo cuáles serían los resultados devueltos por el intérprete.

b) (3 puntos)

Supongamos las siguientes expresiones:

```
(h (g 3) 5)  -> 8
(h (g 12) 5) -> 17
```

Completa las siguientes definiciones de forma que la llamada (g x) devuelva un procedimiento:

```
(define (g x)
  _____)
```

```
(define (h f z)
  _____)
```

## Pregunta 2 (8 puntos)

Define un procedimiento (`calculadora expr`) que tome una lista con la estructura (número op número op... número), donde `-` y `/` son las únicas operaciones permitidas, y calcule el resultado de la operación. Ten en cuenta que la división tiene mayor precedencia que la resta. Por ejemplo,  $(10 - 9 / 3)$  se evalúa como  $10 - (9 / 3)$  y no como  $(10 - 9) / 3$ . Se asume que la lista nunca estará vacía.

Ejemplos:

```
(calculadora '(4 - 2)) -> 2
(calculadora '(10 - 20 / 2 / 5)) -> 8
```

## Pregunta 3 (8 puntos)

Hemos definido la siguientes clases utilizando la extensión de Scheme para POO:

```
(define-class (electrodomestico marca precio)
  (class-vars (cantidad 0))
  (initialize (set! cantidad (1+ cantidad)))
  (method (apagar) ...)
)

(define-class (vitroceramica marca precio)
  (parent (electrodomestico marca precio))
  (method (cambiarPrecio p) (set! precio p))
  (method (enciendeFogon numeroFuego intensidad) ....)
  (method (apagar tiempo)...)
;este método apaga el aparato con un retardo de tiempo minutos
  ...
)
```

a) (3 puntos) Supongamos que se evalúan las siguientes expresiones:

```
(define vitro (instantiate vitroceramica 'aeg 200))
(ask vitro 'cambiarPrecio 500)
```

Se desea obtener el precio original (200) de `vitro`. ¿Cuáles serían las respuestas correctas?

- a) Definiendo un método que devuelva la variable de instanciación `precio`
- b) Almacenando el precio original en una variable de instancia que posteriormente devolveremos
- c) Definiendo el método: `(method (precio-original) (usual 'precio))`
- d) Ninguna de las anteriores

b) (3 puntos) Indica cuales de las siguientes afirmaciones son ciertas y cuales falsas:

- a) Cada instancia de una clase tiene su propia versión de las variables de instanciación
- b) Las variables de instancia se definen como argumentos cuando la instancia se crea
- c) Cuando se envía un mensaje a un objeto para el que no hay definido un

método ejecuta el del hijo si éste existiera

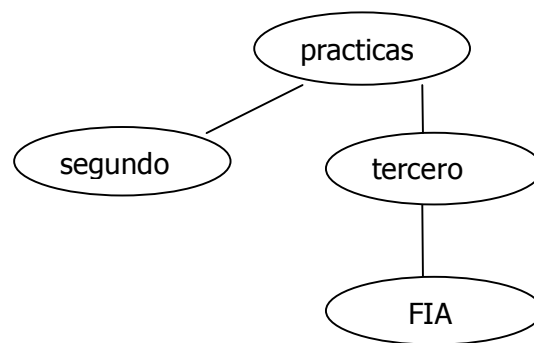
d) La clausula *initialize* permite modificar las variables de instanciación cada vez que se crea una nueva instancia

c) (2 puntos) Se desea agregar un método a la clase *vitrocerámica* para que se apague instantáneamente. Ya dispone de un método con un retardo de x minutos. ¿Qué opciones serían las correctas?

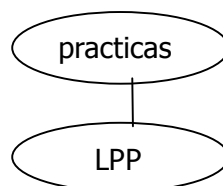
- a) `(method (apagateYa) (ask self 'apagar 0))`
- b) `(method (apagateYa) (ask self 'apagar))`
- c) `(method (apagateYa) (usual 'apagar))`
- d) Las respuestas a) y c) son correctas.

#### Pregunta 4 (9 puntos)

Vamos a representar estructuras de archivos mediante árboles genéricos. Por ejemplo:



a) (5 puntos) Para construir esta estructura, definimos la función *make-rama* que toma una lista con una ruta como argumento y devuelve el árbol que la representa. Por ejemplo, la llamada `(make-rama '(practicas LPP))` debería devolver el árbol siguiente:



El procedimiento *make-rama* podría definirse como:

```
(define (make-rama ruta)
1 (if (null? ruta)
2     '()
3     (make-tree ;; en esta implementación, make-tree = cons
4               (car ruta)
5               (make-rama (cdr ruta)))))
```

Una llamada a `(make-rama '(practicar segundo LPP))` debería devolver `(practicar (segundo (LPP)))`, sin embargo devuelve `(practicar segundo LPP)`. Haz los cambios necesarios para que el procedimiento funcione correctamente. Puede que no sea necesario que cambies todas las líneas.

Cambio línea\_\_\_ por \_\_\_\_\_  
Cambio línea\_\_\_ por \_\_\_\_\_  
Cambio línea\_\_\_ por \_\_\_\_\_  
Cambio línea\_\_\_ por \_\_\_\_\_  
Cambio línea\_\_\_ por \_\_\_\_\_

b) (4 puntos) Define una función `(find-ruta fichero)` que devuelva una lista plana con la ruta en la que se encuentra el fichero. Por ejemplo:

```
(find-ruta 'FIA) -> '(practicar tercero FIA)
(find-ruta 'tercero) -> '(practicar tercero)
(find-ruta 'GRAFICOS) -> '()
```

### Pregunta 5 (8 puntos)

Define el procedimiento `(intercalar! l1 l2)` que reciba dos listas como argumento y, utilizando mutadores, modifique la lista original `l1` e intercale los elementos de ambas. Obviamente, la lista `l2` también quedará modificada.

Ejemplo:

```
(define l1 '(1 2 3 4 5))
(define l2 '(a b c d e))
(intercalar! l1 l2)
l1 -> (1 a 2 b 3 c 4 d 5 e)
l2 -> (a 2 b 3 c 4 d 5 e)
```

### Pregunta 6 (9 puntos)

Supongamos las siguientes expresiones:

```
(define (h z)
  (let ((x z) (y (* 2 z)))
    (lambda (v)
      (set! z v)
      (+ v z x y))))
(define g (h 3))
(g 2)
```

a) (4 puntos) Dibuja el entorno resultante de evaluar las expresiones anteriores

b) (5 puntos) Explica paso a paso cómo se han evaluado las expresiones anteriores