Examen de Lenguajes y Paradigmas de Programación
Convocatoria de Junio de 2002

Nombre y apellidos:

Normas generales

- El examen consta de 2 partes; la primera parte se deberá hacer con el ordenador apagado y en la segunda se podrá usar el ordenador para probar los procedimientos que haya que programar. En ambas partes se pueden usar tanto los apuntes como el libro de la asignatura.
- La nota total de la asignatura será la media entre la nota de prácticas y la nota del examen, siempre que ésta última sea mayor o igual a 3.
- **La duración del examen es de 2 horas**

Parte 1 (sin ordenador)

1. (1 punto) ¿Qué responderá Scheme a las siguientes expresiones?

```
a.- (word 10 (+ 2 3))
b.- (let ((a 5) (b 3)) ((lambda (x) (* a x)) b))
c.- ((lambda (p x) (p 2 x)) * 3)
d.- (append (list 1 2) (cdr (cons 1 nil)))
e.- (car (list (list 1 2) 3))
```

2. (1 punto) Supongamos el siguiente procedimiento

```
(define (multi x)
   (if (null? x)
      (lambda (arg) arg)
      (lambda (arg)
         ((car x) ((multi (cdr x)) arg)))))
```

Explica qué hace multi y escribe un ejemplo de una llamada a la función junto con el resultado que devolvería.

3. (1 punto) Supongamos el siguiente código

```
(define (pairs x lista)
   (if (null? lista) lista
      (append (list (cons x (car lista))) (pairs x (cdr lista)))))
```

Explica qué hace pairs y escribe un ejemplo de una llamada a la función junto con el resultado que devolvería.

4. (1 punto) Dibuja diagramas box-and-pointer que representan los resultados de las siguientes expresiones

```
(define x1 (list (cons 'a 'b) (list 'c 'd)))
(define x2 (cons 'a (cons 'b (cons 'c 'd))))
(define x3 (list 'a 'b 'c (cons 'd '())))
(set-cdr! x3 (cdr x1))
```

Parte 2 (con ordenador)

5. (1,5 puntos) Escribe una función test que tome como argumentos un número y una lista de números y devuelva #t si y sólo si existe alguna pareja de números de la lista que sume lo mismo que el primer argumento.

Por ejemplo:

```
(test 5 (1 4 5 8 3)
#t
(test 5 (2 4 5 8 3)
#f
```

6. (1,5 puntos) Imagina que tienes una versión de Scheme en la que solo están disponibles los números enteros y que, sin embargo, quieres poder representar cantidades de dinero como 3,95 €. Decides crear un tipo abstracto de datos con dos componentes, llamados `euros` y `cents`. Escribes estos constructores y selectores:

```
(define (make-price e c) (+ (* e 100) c))
(define (euros p) (quotient p 100))
(define (cents p) (remainder p 100))
```

(a) Escribe el procedimiento +price que tome dos precios como argumento y devuelva su suma como un nuevo precio. *Respeta la abstracción de datos.*
(b) Ahora queremos cambiar la representación interna de forma que en lugar de representar el precio como un número de céntimos lo representemos como una pareja de dos números. Rescribe los constructores y los selectores necesarios, respetando la abstracción de datos.

7. (1'5 puntos) Recuerda la forma de definir un constructor de contadores en el que se crea una variable "de clase" que mantiene la suma de todos los contadores y una variable "de instancia" que guarda el valor del contador:

```
(define make-count
 (let ((glob 0))
   (lambda ()
     (let ((loc 0))
       (lambda ()
         (set! loc (+ loc 1))
         (set! glob (+ glob 1))
         (list loc glob))))))
```

Escribe una versión modificada de `make-count` en la que la variable del clase `glob` **cuente el número de contadores que se han creado**.

8. (1,5 puntos) Implementa, usando variables locales y paso de mensajes, las funciones de manejo de árboles que hemos visto en teoría:

```
(make-arbol dato hijos)
(dato nodo)
(hijos nodo)
(hoja? nodo)
```

Usando esas funciones, escribe una función (contar-hojas arbol) que cuente el número de hojas de un árbol.