Lenguajes y Paradigmas de Programación
Examen de la convocatoria de Diciembre
Curso 2003-2004

Notas:

1. La duración del examen es de 3 horas
2. Los puntos máximos que se pueden obtener en el examen son 42
3. La puntuación de las prácticas se acumulará a la obtenida en el examen
4. Para aprobar la asignatura es necesario obtener en este examen más de 18 puntos y, junto con las prácticas, más de 30 puntos

Pregunta 1 (6 puntos). Dadas estas definiciones:

```
(define (square x) (* x x))
(define (inc) (set! count (+ count 1)) count)
(define count 5)
```

¿Cuál es el valor de la expresión `(square (inc))` bajo un orden de evaluación aplicativo? ¿Cuál esl el valor de la misma expresión bajo un orden de evaluación normal?

Pregunta 2 (6 puntos). Escribe una función prefijo-a-infijo que tome una expresión aritmética de Scheme como argumento, en forma de árbol infijo, y devuelva una lista con la expresión en la forma de notación usual infija, con los operadores entre los operandos, como esto:

```
> (prefix-to-infix '(+ (* 2 3) (- 7 4)))
((2 * 3) + (7 - 4))
> (prefix-to-infix '(* (remainder 9 2) 5)
((9 remainder 2) * 5)
```

Pregunta 3 (10 puntos). Esta pregunta se refiere al tipo abstracto de datos árbol, definido por estos selectores y constructor:

```
(define datum car)
(define children cdr)
(define make-node cons)
```

Escribe el procedimiento de alto nivel `tree-accumulate`. Sus dos argumentos son una función `fn` y un árbol. La función `fn` tomará dos argumentos y será asociativa (esto es, no tienes que preocuparte del orden en el que encuentres los nodos en el árbol). El procedimiento `tree-accumulate` combinará todos los datos del árbol mediante la aplicación de `fn` a ellos de dos en dos, de forma análoga al procedimiento ordinario accumulate. Por ejemplo:

```
> (define my-tree
    (make-node 3 (list (make-node 4 '())
                       (make-node 7 '())
                       (make-node 2 (list (make-node 3 '())
                                          (make-node 8 '()))))))
> (tree-accumulate + my-tree)
27
> (tree-accumulate max my-tree)
8
```

Pregunta 4 (10 puntos). Escribe un procedimiento deep-subst! que tome tres argumentos, dos de los cuales son palabras y el tercero es cualquier estructura de lista (cualquier cosa hecha de parejas). El procedimiento debe mutar la estructura de la lista de forma que cualquier ocurrencia de la primer palabra se substituya por la segunda palabra. Ejemplos:

```
> (deep-subst! 'foo 'baz (list (cons 'hello 'goodbye) (cons 'moby 'foo)))
((hello . goodbye) (moby . baz))

> (deep-subst! 'a 'x (list (list 'a 'b 'c) (list 'b 'a 'd)
                           (list 'f 'a 'b)))
((x b c) (b x d) (f x b))
```

La función no debe crear nuevas parejas.

Pregunta 5 (10 puntos). Dibuja los diagramas de entornos resultantes de la evaluación de las siguientes expresiones:

```
(define (kons a b)
  (lambda (m)
    (if (eq? m 'kar) a b)))
(define p (kons (kons 1 2) 3))
```