Lenguajes y Paradigmas de Programación
Curso 2005-2006
Examen de la Convocatoria de Septiembre

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 viernes 15 de Septiembre.

Pregunta 1 (10 puntos)

Escribe un procedimiento `(diferencias-mayor-que lista n)` que tome una lista con números y un número `n` como argumento. Deberá devolver el número de veces que el valor absoluto de la diferencia entre dos números consecutivos en la lista es mayor que el número `n`. Explica si tu procedimiento es recursivo o iterativo.

Ejemplos:

```
(diferencias-mayor-que '(1 0 4 8 9 0 6) 2)
4
(diferencias-mayor-que '(2 4 6 8 10) 3)
0
```

Pregunta 2 (10 puntos)

Supongamos n estudiantes haciendo un examen de LPP. Podemos representar la nota de cada estudiante en cada elemento de una lista (de longitud n). Escribe un procedimiento llamado `(histograma notas)` que tome una lista de notas como parámetro y devuelva su histograma. Es decir, la lista resultante deberá ser de longitud M, donde M es la máxima puntuación obtenida por un alumno en el examen y cada elemento i-ésimo de la lista representa el número de estudiantes que han obtenido la puntuación i en el examen.

Ejemplos:

```
(histograma '(3 2 2 3 2))
(0 0 3 2) ;; ningún estudiante obtuvo 0 puntos
          ;; ningún estudiante obtuvo 1 punto
          ;; 3 estudiantes obtuvieron 2 puntos
          ;; 2 estudiantes obtuvieron 3 puntos
```

```
(histograma '(0 1 0 2))
(2 1 1) ;; 2 estudiantes obtuvieron 0 puntos
        ;; 1 estudiante obtuvo 1 punto
        ;; 1 estudiante obtuvo 2 puntos
```

Pista: te puede ser de ayuda comenzar por implementar la función auxiliar `(inc-nth lista n)` que incrementa en 1 la posición n-ésima de la lista, o añade ceros a la derecha (seguidos de un 1) si la posición n es mayor o igual que la longitud de la lista:

```
(define lista '(0 0 0))
(inc-nth lista 0) ; lista = (1 0 0)
(inc-nth lista 1) ; lista = (1 1 0)
(inc-nth lista 0) ; lista = (2 1 0)
(inc-nth lista 4) ; lista = (2 1 0 0 1)
lista
(2 1 0 0 1)
```

Para implementar esta función puedes usar la función de Scheme `(list-tail lista n)` que devuelve la pareja n-ésima de la lista (o sea, la sublista que comienza en la posición `n`) y debes usar algún mutador para modificar la lista sin tener que crear parejas nuevas.

Ejemplos de `list-tail`:

```
(list-tail '(1 2 3 4) 1) -> (2 3 4)
(list-tail '(1 2 3 4) 4) -> ()
(list-tail '(1 2 3 4) 0) -> (1 2 3 4)
(list-tail '(1 2 3 4) 5) -> error!!
```

Pregunta 3 (10 puntos)

Supongamos este programa para ordenar listas:

```
(define (sort lst)
   (if (null? lst)
      '()
      (insert (car lst) (sort (cdr lst)))))

(define (insert value sorted)
   (cond ((null? sorted) (list value))
         ((< value (car sorted)) (cons value sorted))
         (else (cons (car sorted)
                     (insert value (cdr sorted))))))
```

Vamos a reescribirlo utilizando mutación, reordenando el orden de las parejas y sin llamar a `cons` para crear nuevas parejas:

```
(define (sort! lst)
   (if (null? lst) '()
      (insert! lst (sort! (cdr lst)))))

(define (insert! value-pair sorted)
   (cond ((null? sorted)
            (set-cdr! value-pair '())
            value-pair)
          ((< (car value-pair) (car sorted))
              ...)
          (else
              ...)))
```

Ejemplo de uso:

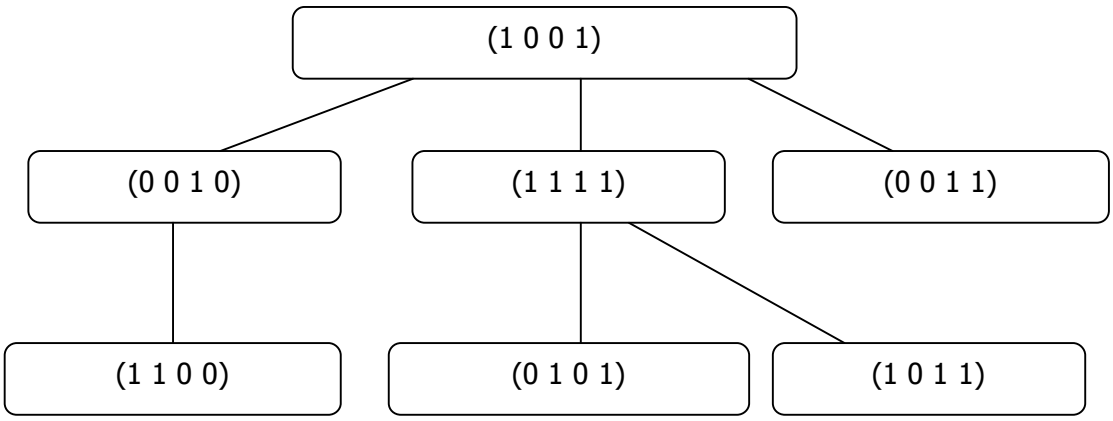
```
(sort! (list 7 3 87 5))
(3 5 7 87)
```

a) **(7 puntos)** Completa la definición de `insert!`.

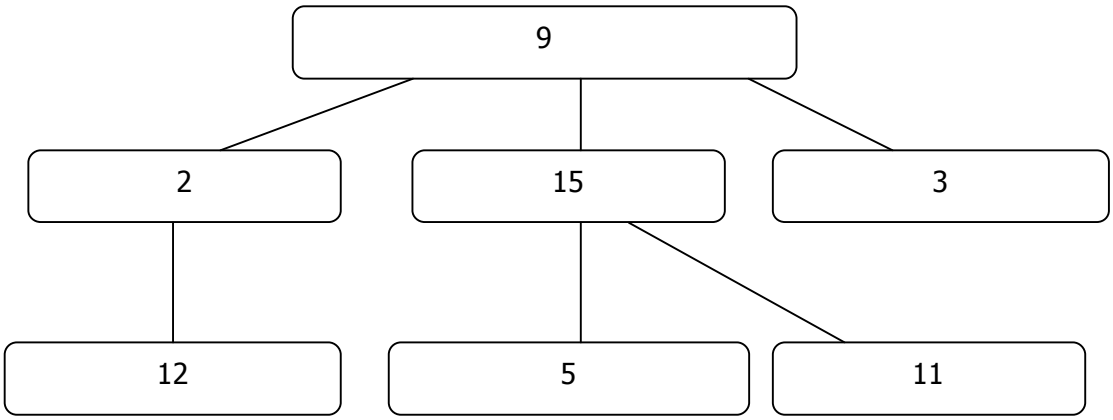
b) **(3 puntos)** Una vez implementado `sort!` e `insert!` te das cuenta de que tal y como están definidos sólo se pueden aplicar a ordenar listas de números. ¿Cómo podrías definirlos de una forma más genérica, para que se pudiera aplicar a cualquier tipo de dato para el que existiera una relación de orden (relación que indica si un dato es menor, mayor o igual que otro).? Explícalo, indicando cómo cambiar las funciones `sort!` e `insert!` y pon un ejemplo.

Pregunta 4 (10 puntos)

Un compañero está haciendo una práctica en la que utiliza árboles genéricos que contienen números binarios del 0 al 15 (4 bits). Para representar los números binarios utiliza listas de 4 elementos. De esta forma, su programa maneja árboles como el siguiente:



Preocupado por el consumo de memoria de esta representación, le comentas que sería mejor que representara los números binarios utilizando enteros decimales, con lo que quedaría un árbol como el siguiente:



a) (3 puntos) Implementa las funciones `(lista-binaria-a-numero lista)` y `(numero-a-lista-binaria numero)` que transformen un tipo de datos en otro.

Pista: Recuerda que el valor decimal de un número binario (a_3 a_2 a_1 a_0) es $a_3*2^3+a_2*2^2+a_1*2^1+a_0*2^0$.

b) (5 puntos) Utilizando las funciones anteriores, implementa los procedimientos `(reduce-tree tree)` y `(amplia-tree tree)` que reciba un árbol genérico como argumento y devuelva el árbol con el tipo de dato reducido (número) o ampliado (lista binaria).

c) (2 puntos) Una vez que has implementado las funciones anteriores, te das cuenta de que podríamos mejorar la abstracción si simplemente implementáramos una función `(transforma-tree tree)` que ampliara el árbol si está reducido y lo redujese si está ampliado. Explica cómo implementarías esta función.

Pregunta 5 (10 puntos)

Supongamos las siguientes expresiones:

```
(define (sumador-x)
   (lambda (y)
      (+ x y)))
(define x 10)
(define foo
   (let ((x 5))
      (sumador-x)))
(foo 3)
```

a) (2 puntos) ¿Qué valor devuelve la última expresión?
b) (5 puntos) Dibuja el diagrama de entornos resultante de evaluar las expresiones.
c) (3 puntos) Explica la evaluación de las expresiones usando el modelo de entorno.