Lenguajes y Paradigmas de Programación
Curso 2003-2004
Examen de la Convocatoria de Junio

Normas importantes

- La puntuación total del examen es de 40 puntos que sumados a los 20 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 16 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.

Pregunta 1 (5 puntos)

Queremos un procedimiento (`haz-palindromo wd`) que tome una palabra `wd` y que devuelva un palíndromo formando a partir de la palabra `wd`. Escribe el procedimiento e indica si es recursivo o iterativo.

Ejemplos:

```
(haz-palindromo "hola") -> "holaaloh"
(haz-palindromo "arroz")-> "arrozzorra"
(haz-palindromo "") -> ""
```

Pregunta 2 (5 puntos)

Rellena los huecos en las siguientes expresiones de Scheme para que su evaluación devuelva los resultados indicados. No es necesario que en los huecos haya un único dato, puede haber una expresión compuesta.

a)

```
(define g
   (lambda (______)
      (lambda (______ b) (______ b))

((g 3) - 5)
```

-2

b)

```
(define object
   (let ((init 0))
      (______ (new)
         (let ((temp init))
            (______) temp))))

(object 7)
0
(object 1)
7
(object 4)
1
```

c) Indica cuál es el resultado de evaluar las siguientes expresiones de Scheme.

```
(define (echo f n)
   (if (= n 0)
      (lambda (x) x)
      (lambda (x) (f ((echo f (- n 1)) x)))))

((echo (lambda (x) (* x x)) 2) 2)
___
```

Pregunta 3 (6 puntos)

Escribe un procedimiento `(producto-diagonal matriz)` que calcule el producto de la diagonal principal de una matriz cuadrada `matriz`. Suponemos que la matriz cuadrada se representa como una lista de listas planas en la que cada lista plana representa una fila de la matriz. Puedes definir procedimientos auxiliares que uses en el procedimiento principal.

Ejemplos:

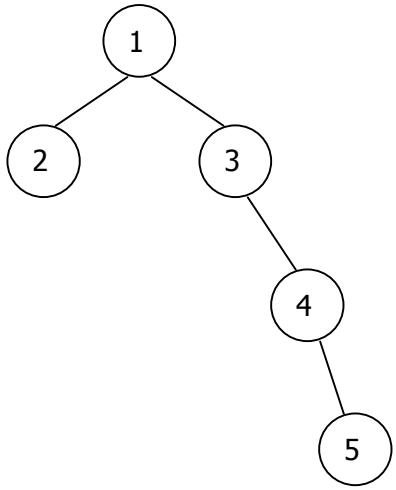
```
(producto-diagonal '((2 3)
                     (4 5)))
10

(producto-diagonal '((2   4  6)
                     (8  10 12)
                     (14 16 18)))
360
```

Pregunta 4 (6 puntos)

Define el procedimiento `(max-hijos tree)` que vaya recorriendo los nodos del árbol `tree`, calculando el número de hijos de cada nodo y que devuelva el máximo número de hijos encontrado en un único nodo.

Por ejemplo, el árbol `'(1 (2) (3 (4 (5))))` se corresponde con la siguiente figura:



En este árbol tenemos los siguientes números de hijos:

Nodo ‘1’ tiene 2 hijos
Nodo ‘2’ tiene 0 hijos
Nodo ‘3’ tiene 1 hijo
Nodo ‘4’ tiene 1 hijo
Nodo ‘5’ tiene 0 hijos

Por lo tanto, el máximo número de hijos corresponde al nodo ‘1’, con 2 hijos. Éste número es el que debe devolverse.

Ejemplos:

```
(max-hijos '(1 (2) (3) (4)))
3
(max-hijos '(1 (2) (3 (4 (5)))))
2
```

Pregunta 5 (6 puntos)

Queremos definir con la extensión de Programación Orientada a Objetos de Scheme una jerarquía de clases que nos permita mantener la información sobre el reparto de paquetes de una empresa de mensajería a contra-reembolso.

Definimos una clase llamada **`Paquete`** que tendrá como variables de instanciación el **nombre del mensajero** que se encargará de su reparto y el **precio del objeto** a entregar.
Además se guardará la información del estado en que se encuentra un paquete. Existirán tres posibles valores:

0: perfecto
1: leve deterioro
2: deteriorado

Por último, existen dos tipos distintos de paquetes: **`Sobre`** y **`Caja`**. El transporte de un sobre tiene un **coste de transporte** de 2 euros y el de una caja de 6 euros siempre que el paquete se entregue en estado perfecto.

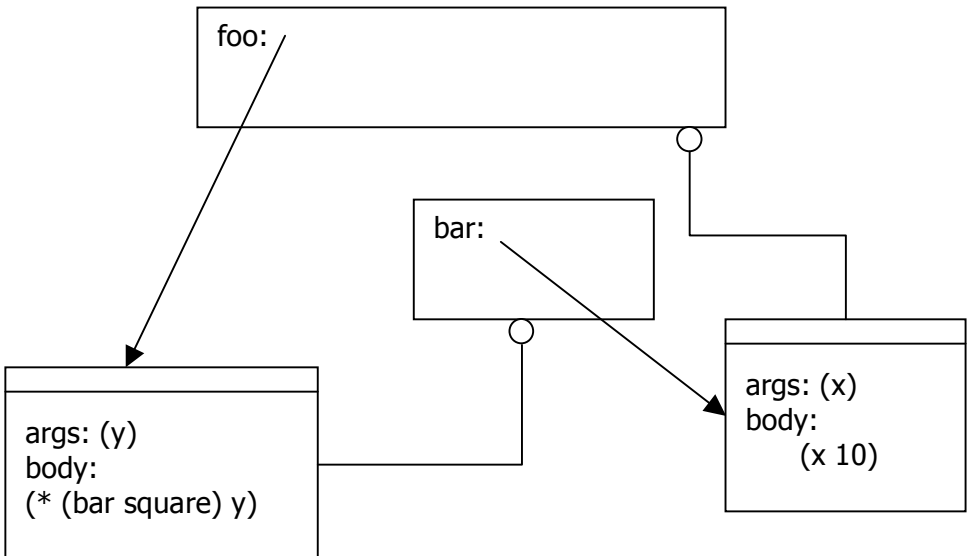
Necesitamos los siguientes métodos:

- `num-paquetes`, que devuelve el número de paquetes creados
- `num-sobres`, que devuelve el número de sobres
- `num-cajas`, que devuelve el número de cajas
- `cambia-mensajero`, que permite cambiar el mensajero asignado a un paquete
- `cambia-estado`, que permite cambiar el estado de un paquete. Cuando un paquete está deteriorado ese será su estado definitivo y el estado no se podrá modificar.
- `importe`, que devuelve el precio a cobrar por la entrega de un paquete. El precio a cobrar es el coste de transporte más el precio del objeto, si el estado es 0 (perfecto). Si el estado es 1, el coste de transporte se reduce en un 50% y si el estado es 2 el coste de transporte es 0 y el importe a cobrar es el precio del objeto.

Usando la extensión de POO de Scheme, implementa un conjunto de clases que resuelvan este problema y escribe un ejemplo de uso.

Pregunta 6 (6 puntos)

Escribe **una única instrucción** en Scheme, que no use mutación, que genere el siguiente modelo de entorno:



Pregunta 7 (6 puntos)

Un árbol binario infinito es una estructura que contiene un dato, un hijo a la izquierda y un hijo a la derecha, los cuales son a su vez árboles binarios infinitos. Es decir, un árbol binario infinito es un árbol cuyos nodos tienen dos hijos y un profundidad infinita (infinitos niveles).

a) Rellena los huecos para completar la definición del árbol binario infinito y de sus selectores. Supongamos que la llamada a `def-macro` en la siguiente expresión permite definir una nueva macro de Scheme. En este caso es necesario definir `make-inf-tree` como una macro y no como una función porque, de forma similar a lo que sucede con los streams, *no queremos que se evalúen los argumentos* cuando se haga la llamada a `make-inf-tree` para construir el árbol infinito.

```
(def-macro (make-inf-tree dato izq der)
    (list dato ______))

(define (dato tree) (______))
(define (hijo-izq tree) (______))
(define (hijo-der tree) (______))
```

b) Utilizando la estructura de árbol binario infinito, define el procedimiento `(make-n-tree n)` que reciba un número `n` como parámetro y devuelva el siguiente árbol binario infinito: