Lenguajes y Paradigmas de Programación
Curso 2002-2003
Examen de la Convocatoria de Junio

Normas importantes

- La puntuación total del examen son 42 puntos que sumados a los 18 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 más de 18 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 (3 puntos)

Escribe un procedimiento `add-numbers` que tome una frase como argumento. Algunas de las palabras en la frase pueden ser números. El procedimiento deberá devolver la suma de estos números, por ejemplo:

```
>(add-numbers '(8 dias 1 semana))
9

>(add-numbers '(76 trombones 4 flautas y 2 trompetas))
82

>(add-numbers '(all you need is love))
0
```

Pregunta 2 (5 puntos)

Rellena los espacios en blanco para obtener el resultado esperado:

a) (2 puntos)

```
>((lambda (x y z) (x (_______) z)) * + 3)
> 15
```

b) (3 puntos)

```
>(((lambda (a b) b) + ____) 3 5)
> 15
```

Pregunta 3 (4 puntos)

Escribe un procedimiento `(pref wd char)` que tomo como argumentos una palabra `wd` y un carácter `char` y que devuelva el prefijo de la palabra `wd` hasta llegar al carácter char.

Ejemplo:

```
> (pref "hola" "l")
> "ho"
> (pref "pepito" "i")
> "pep"
> (pref "hola" "i")
> "hola"
```

El procedimiento que has escrito es ¿recursivo o iterativo?. Explica la respuesta.

Pregunta 4 (3 puntos)

Vamos a crear un TAD para representar una fecha. El constructor `(make-fecha dia mes año)` tendrá tres argumentos: día mes año.
Tendremos cuatro operadores:

> `(año fec)`-> devuelve el año de la fecha pasada por parámetro.
> `(mes fec)`-> devuelve el mes de la fecha pasada por parámetro.
> `(dia fec)`-> devuelve el dia de la fecha pasada por parámetro.
> `(fin-mes fec)`-> devuelve true si es el último día del mes de la fecha fec (no tener en cuenta años bisiestos).

- Implementar `make-fecha`, `año`, `mes`, `dia` y `fin-mes` usando una representación interna de una lista de tres números. Es decir 02/12/2001 (02 12 2001).
- Implementar `make-fecha`, `año`, `mes`, `dia` y `fin-mes` usando una representación interna en la que la fecha se representa como un entero ((añox10000)+(mesx 100)+dia)

Pregunta 5 (5 puntos)

Escribe una función `(max-levels)` que recorra una lista de listas (un árbol sin datos en los nodos) y que devuelva el número de niveles de la rama que tiene mayor profundidad.

Por ejemplo

```
> (max-levels '(1 2 3 4))
> 1
> (max-levels '(1 (2) 3))
> 2
> (max-levels '((1) (2 (3))))
> 3
```

Pregunta 6 (4 puntos)

Vamos a hacer un conversor de monedas. Imaginariamente podemos tener una progresión lineal con las conversiones de moneda de este tipo:

... -> dólar -> rupia -> euro -> peseta -> ...
... -> 1 -> 12 -> 62 -> 166 -> ...

Hemos rellenado una tabla con las conversiones:

```
(put 'convert 'dolar (attach-tag 'rupia 12))  ; 12 rupias son 1 dolar
(put 'convert 'rupia (attach-tag 'euro 62))   ;  62 euros son 1 rupia
(put 'convert 'euro (attach-tag 'peseta 166)) ; 166 pesetas son 1 euro
....
```

Sólo hay una conversión para cada moneda. Si se quiere convertir de una moneda a otra que no sea correlativa, se deberá convertir una a una hasta llegar al tipo deseado. Por ejemplo, si queremos pasar de rupias a pesetas, debemos convertir rupias a euros y euros a pesetas.

Escribe el procedimiento `conversion` que tenga dos argumentos, `desde` y `hasta`, y deberá devolver un número que represente la conversión de la moneda `desde` a la moneda `hasta`.

```
>(conversion 'dolar 'rupia)
>12
>(conversion 'dolar 'euro)
>744
```

Pregunta 7 (7 puntos)

Queremos definir en POO una clase llamada **animal**, que tendrá como variable las vidas que tiene un animal y que por defecto valdrá 1. También definiremos las clases **gato** y **perro**, teniendo en cuenta que una gato tiene 7 vidas. Implementar un método que nos devuelva la cantidad de animales que se han creado, otro que nos devuelva la cantidad de perros y otro para la cantidad de gatos. Además implementaremos un método que se llamará **finVida** que restará una vida al animal al que se le aplique, de forma que cuando no le queden más vidas visualizará un cero. También implementaremos un método **estado** que devuelva el número de vidas de un animal. Por último podremos jugar a ser *"DIOS"* y redefinir el número de vidas de los animales que queramos, con el método **cambioVidas**. Este método cambia el contador de vidas del animal al número que queramos, siempre que ese número no sea mayor que la cantidad máxima de vidas que tiene el tipo de animal (1 para perro y 7 para gato).

Pregunta 8 (6 puntos)

Explica la evaluación de las siguientes expresiones y dibuja el entorno resultante.

```
(define m
   (lambda (x)
      (lambda (y)
         (+ x y))))

(define k (m 5))

(k 7)
```

Pregunta 9 (5 puntos)

Queremos implementar el procedimiento `swap-cars!` que tome dos listas como parámetros e intercambie entre ellas el primer elemento de cada una. Por ejemplo:

```
>(define pares (list 2 4 6))
>(define impares (list 1 3 5))
>(swap-cars! pares impares)
>pares
>(1 4 6)
>impares
>(2 3 5)
```

(a) (3 puntos) ¿Cuál o cuáles de los siguientes procedimientos funcionan correctamente? Rodea SI o NO en cada uno. Si rodeas el NO, razona por qué en la casilla de la derecha.

SI
NO
	`(define (swap-cars! x y)`
	`  (let ((temp x))`
	`  (set! x (cons (car y) (cdr x)))`
	`  (set! y (cons (car temp) (cdr y`
	`)))))`	

SI
NO
	`(define (swap-cars! x y)`
	`  (let ((temp (car x)))`
	`  (set! (car x) (car y))`
	`  (set! (car y) temp)))`	

SI
NO
	`(define (swap-cars! x y)`
	`  (set-car! x (car y))`
	`  (set-car! y (car x)))`	

SI
NO
	`(define (swap-cars! x y)`
	`  (let ((temp (cons (car x) (cdr x))))`
	`  (set-car! x (car y))`
	`  (set-car! y (car temp))))`	

(b) (2 puntos) Rellena el espacio en blanco del siguiente procedimiento para que swap-cars! funcione correctamente:

```
(define (swap-cars! X y)
      (set! x (cons (car y) x))
      (set-car! y (cadr x))
      ;; Rellena el espacio en blanco con una
      ;; llamada a set!, set-car! o set-cdr!

      (_____________________________))
```