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

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 día 1 de Julio.

Pregunta 1 (8 puntos)

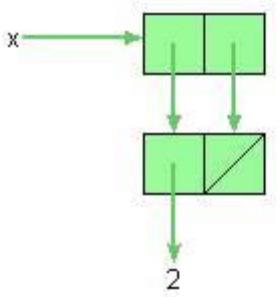
Define un procedimiento llamado `(transformar plantilla lista)` que reciba dos listas como argumento: la lista `plantilla` estará compuesta por números enteros positivos con una estructura jerárquica, como `(2 (3 1) 0 (4))`. La segunda `lista` será una lista plana con tantos elementos como indica el mayor número de `plantilla` más uno (en el caso anterior serían 5 elementos). El procedimiento deberá devolver una lista donde los elementos de la segunda lista se sitúen (en situación y en estructura) según indique la `plantilla`.

Ejemplos:

```
(transformar '((0 1) 4 (2 3)) '(hola que tal estas hoy))
((hola que) hoy (tal estas))

(transformar '(1 4 3 2 5 (0)) '(vamos todos a aprobar este examen))
(todos este aprobar a examen (vamos))
```

Pregunta 2 (8 puntos)

a) **(4 puntos)** Tenemos el siguiente box-and-pointer:



Para cada una de las siguientes expresiones que lo gene
correctas y cuáles no. En caso de que se incorrecta, debes explicar por qué.

Expresión	Verdadero / Falso (indicar por qué)
`(define x '((2) 2))`	

`(define x`
`(let ((temp (list 2)))`
`(cons temp temp)))`	

`(define x`
`(let ((temp (list 2)))`
`(let ((foo (cons (list 2) temp)))`
`foo)))`	

`(define x`
`(let ((temp (list 2)))`
`(let ((foo (cons (list 2) temp)))`
`(set-car! foo temp)`
`foo)))`	

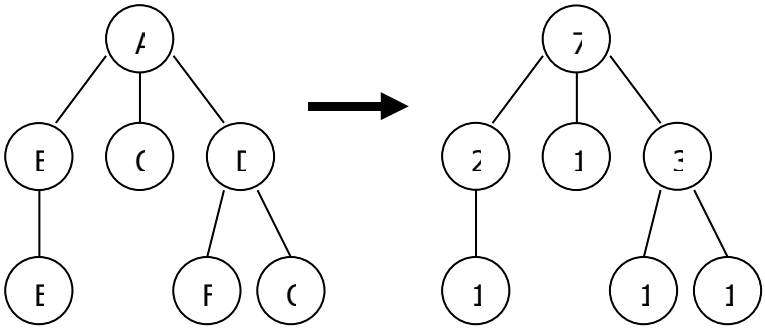
`(define x`
`(let ((temp (list 2)))`
`(let ((foo (cons (list 2) temp)))`
`(set! (car foo) temp)`
`foo)))`	

b) **(4 puntos)** El siguiente programa tiene errores. Corrígelos e indica cuáles son y el tipo de error (los errores no son sintácticos ni están en el número de paréntesis)

```
1. (define attach-type cons)
2. (define type car)
3. (define contents cdr)

4. (define (make-euro amt)
5.   (attach-type 'euro amt))

6. (define (make-dolar amt)
7.   (attach-type 'dolar amt))

8. (define (make-libra amt)
9.   (attach-type 'libra amt))

10. (define (+money amt1 amt2)
11.   (make-euro (+ (contents (toeuro amt1))
12.                 (contents (toeuro amt2)))))

13. (define (toeuro amt)
14.   (* (conversion (car amt)) (cdr amt)))

15. (define (conversion type)
16.   (cond ((eq? type 'euro) 1.0)
17.         ((eq? type 'dolar) 0.7)
18.         ((eq? type 'libra) 1.2)))
```

Pregunta 3 (9 puntos)

Define un procedimiento llamado `(set-nodes! tree)` que reciba un árbol genérico como argumento y mute el contenido de los nodos del árbol de forma que cada uno contenga el número de nodos del subárbol del que es raiz. Ejemplo:



Pregunta 4 (9 puntos)

Recuerda la función **`map`** que toma como argumento una función y una lista y devuelve la lista resultante de aplicar la función a cada elemento de la lista. Veamos una variante, la función **`mapfun`**, que toma una lista de funciones y un valor y devuelve una lista resultante de aplicar cada una de las funciones al valor. Por ejemplo:

```
(mapfun (list (lambda (x) (+ x 3)) (lambda (x) (+ x 10))) 5) -> (8 15)
```

Supongamos ahora que llamamos **`suma-`*`k`*** a la función de un parámetro que suma el número ***`k`*** a ese parámetro. Por ejemplo **`suma-5`** sería una función que le suma 5 al parámetro que se le pasa:

```
(suma-5 10) -> 15
```

Definimos entonces la función **`(sumadores n k)`** como la función que devuelve una lista de n funciones sumadoras que suman los números k, 2k, …, nk (esto es, la lista de funciones **`suma-k, suma-2k, …, suma-nk`**). Por ejemplo, **`(sumadores 4 3)`** devuelve una lista con las funciones **`suma-3`**, **`suma-6`**, **`suma-9`** y **`suma-12`**. De esta forma, si unimos la llamada a **`mapfun`** con lo que nos devuelve **`sumadores`**, obtendremos el siguiente ejemplo:

```
(mapfun (sumadores 4 3) 10) -> (13 16 19 22)
```

a) **(4 puntos)** Implementa la función **`mapfun`**
b) **(5 puntos)** Implementa la función **`sumadores`**

Pregunta 5 (8 puntos)

Supongamos las siguientes expresiones:

```
(define (f y)
   (let ((x 10))
      (lambda (y)
         (set! x (+ x y))
         x)))
(define x 2)
```

```
(define g (f (+ x 3)))
(g x)
```

a) **(4 puntos)** Dibuja el entorno resultante de evaluar las expresiones anteriores
b) **(4 puntos)** Explica paso a paso cómo se han evaluado las expresiones anteriores

Pregunta 6 (8 puntos)

Lee el siguiente problema a programar:

> **Explorando las minas de Moria.**
> Supongamos una aventura en la que un personaje puede moverse por las minas de Moira. Las minas de Moira son un conjunto de estancias comunicadas entre si. Cada estancia tiene un identificador (su nombre). Las estancias se conectan entre si mediante puertas situadas en el norte, sur, este u oeste (una estancia puede tener un número variable de puertas: de una a cuatro). Cada puerta tiene un identificador único.
>
> El personaje podrá moverse de una habitación a otra realizando uno de los siguientes cuatro movimientos: ir-norte, ir-sur, ir-este e ir-oeste. Si la puerta correspondiente existe y está abierta en la estancia actual, el personaje pasará a la estancia situada en esa posición. Si la puerta no existe o está cerrada, el persona permanecerá en la misma habitación.
>
> En las habitaciones puede haber distintos objetos, entre ellos llaves. Todos los objetos tienen un identificador que los distingue. Los identificadores de las llaves coincidirán con un identificador de alguna puerta, en cuyo caso la llave puede abrir la puerta. El personaje puede coger los objetos y guardárselos (el objeto deja entonces de estar en la habitación). Si el objeto es una llave, podrá usarla para abrir y cerrar la puerta correspondiente a su identificador.

Una vez leído el problema, contesta a las siguientes cuestiones:

a) **(4 puntos)** Diseña un conjunto de **clases**, **interfaces** y **relaciones de herencia** con las que especificar el problema. Para cada clase debes definir sus campos y sus métodos, explicando qué hace cada uno de ellos **pero sin implementarlos** (si quieres alguna pista de posibles clases y métodos, sigue leyendo). Puedes dibujar diagramas UML como los vistos en clase (pero explicando además cada componente).

b) **(2 puntos)** Implementa en la clase **`personaje`** el método **`(abre-puerta dirección)`** (siendo **`dirección`** uno de los valores: **`´norte`**, **`´sur`**, **`´este`**, **`´oeste`**) que pruebe a abrir la puerta correspondiente de la estancia actual con todas las llaves que lleva el personaje. Debes usar siempre métodos que hayas definido en el apartado anterior.

c) **(2 puntos)** Implementa en la clase **`personaje`** el método **`(coge-objeto identificador)`** que realiza la acción de coger de la estancia actual el objeto especificado por el identificador.