Lenguajes y Paradigmas de Programación
Curso 2006-2007
Examen de la Convocatoria de Junio

Normas importantes

- La puntuación total del examen es de 60 puntos. Para obtener la nota final (en escala 0-10) se suman los puntos de prácticas y se divide por 6.
- Para sumar los puntos de las prácticas **es necesario obtener un mínimo de 24 puntos en este examen**.
- Se debe contestar cada pregunta **en una hoja distinta**. No olvides poner el nombre en todas las hojas.
- La duración del examen es de 3 horas.
- Las notas (y la fecha de revisión) estarán disponibles en la web de la asignatura el próximo día 2 de Julio.

Pregunta 1 (5 puntos)

Contesta a las siguientes preguntas, rodeando la respuesta que consideres correcta (sólo una es cierta). Cada pregunta vale 1 punto.

1. En programación declarativa ¿es posible definir una referencia?

- (a) Depende del lenguaje de programación.
- (b) No, porque en programación declarativa las variables sólo son nombres que representan valores, similares a constantes.
- (c) No, porque la programación declarativa está basada en la declaración de funciones.
- (d) Sí, por ejemplo cuando damos valor a un parámetro de una función.

2. Un modelo de computación proporciona:

- (a) Una explicación del funcionamiento de los programas de un lenguaje de programación.
- (b) Una forma de determinar si un programa es correcto o no.
- (c) Unas reglas que analizan sintácticamente un programa.
- (d) Un mecanismo de ejecución virtual de cualquier lenguaje de programación de un paradigma.

3. En el modelo de computación basado en entornos, se crea un entorno nuevo:

- (a) Cada vez que se llama a una función primitiva.
- (b) Cada vez que se llama a un procedimiento definido por el usuario.
- (c) Cada vez que se llama a un “define”.
- (d) Cada vez que se llama a un “lambda”.

4. Una de las ventajas principales de la utilización de barreras de abstracción es que:

- (a) Los programas son más rápidos.
- (b) Los programas se pueden ejecutar en distintos sistemas operativos.

- (c) Se oculta la implementación y se acerca el lenguaje de programación al dominio que se está programando.
- (d) El funcionamiento de los programas se puede comprobar automáticamente.

5. Una de las ventajas principales de los lenguajes fuertemente tipeados es que:

- (a) Los entornos de programación pueden detectar más errores mientras el programador va escribiendo el programa.
- (b) Los programas son más eficientes.
- (c) Es posible definir polimorfismo.
- (d) Una función puede devolver más de un tipo de datos.

Pregunta 2 (5 puntos)

1. Dadas la siguiente expresión:
   a) Dibuja el diagrama *box & pointer* correspondiente.
   b) Escribe la salida por pantalla que produce Scheme después de la evaluación.

```
(define z
    (let ((y '(7)))
        (let ((y '(8))
              (x (list 'a '(b c) (car y))))
           (set-car! x (car y))
           (set-cdr! x (car (cdr x)))
            x)))
```

Pregunta 3 (10 puntos)

a) Implementa la función `(split lista n)` que divide una lista en dos por el elemento n-ésimo **sin utilizar mutadores**. El resultado es una pareja cuyo `car` es la primera sublista y cuyo `cdr` es la segunda. Ejemplo:

```
(define lista '(1 2 3 4 5))
(define pareja (split lista 3)
(car pareja) -> '(1 2 3)
(cdr pareja) -> '(4 5)
lista -> '(1 2 3 4 5)
```

b) Implementa la función `(split! lista n)` que haga lo mismo que `split` pero **utilizando mutadores**. Ejemplo:

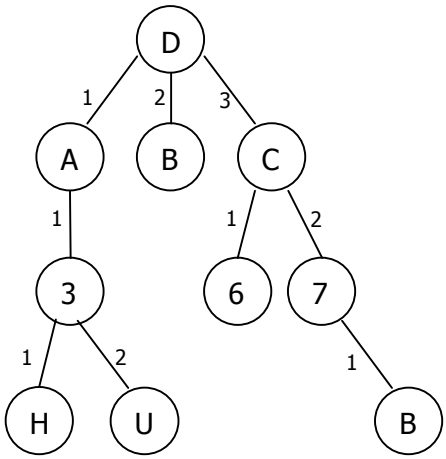
```
(define lista '(1 2 3 4 5))
(define pareja (split lista 3)
(car pareja) -> '(1 2 3)
(cdr pareja) -> '(4 5)
lista -> '(1 2 3)
```

Pregunta 4 (10 puntos)

Queremos un procedimiento que nos permita acceder de forma indexada a los elementos de un árbol.

Escribe un procedimiento `tree-ref` que tome un árbol genérico y una lista de índices (números enteros mayores o igual a 1) como argumentos. La secuencia de índices muestra el **camino** a seguir para encontrar un elemento en el árbol.

Por ejemplo, considerando el árbol que se muestra a continuación, la llamada a `(tree-ref tree '(3 1))` devuelve 6. El primer índice, el 3, selecciona el subárbol que se encuentra en la tercera rama del árbol, y el índice 1 selecciona el subárbol que está en la primera rama de ese subárbol. Devuelve el dato que se encuentra en el subárbol indexado (el 6).



Más ejemplos:

```
(tree-ref tree '(1 1 1))
H
(tree-ref tree '(3 2))
7
(tree-ref tree '(1))
A
```

Suponemos que la secuencia de índices siempre es correcta, es decir, proporciona un camino existente en el árbol.

Pregunta 5 (10 puntos)

Supongamos que necesitamos mantener simultáneamente dos implementaciones del tipo de dato `rectangulo2D`, que llamaremos `rect-base-altura` y `rect-coords`.

En la primera implementación representamos un rectángulo por la posición (x e y) de su esquina inferior izquierda y por su base y altura.

En la segunda implementación lo representamos mediante la posición de su esquina inferior izquierda y de su esquina superior derecha.

a) (5 puntos) Define la barrera de abstracción de los tipos específicos `rect-base-altura` y `rect-coords` (un mínimo de 3 funciones), indicando el nombre de cada función, sus argumentos y su comportamiento. Define una barrera de abstracción común con un mínimo de 3 funciones genéricas (polimórficas) que admitan cualquier tipo específico de rectángulo, describiendo su comportamiento.

b) Define una implementación completa de las funciones anteriores, utilizando el enfoque de paso de mensajes.

Pregunta 6 (10 puntos)

Estamos diseñando una aventura mediante PDD. Se trata de un laberinto formado por un conjunto de habitaciones unidas por pasadizos. Un ejemplo de utilización (en negrita aparece lo que el jugador escribe por teclado) sería:

```
(aventura)
Te encuentras en la entrada de una cueva. Hay un túnel hacia el este.
Movimiento? este
Te encuentras en una cueva amplia. Las paredes brillan. Hay túneles al
este y al sur. Hay una varita mágica en el suelo.
Movimiento? coger varita
Aparece una nube de humo. Te encuentras ahora en una habitación con
paredes de oro. Un frasco con una poción verde está en el suelo. Hay
unas puertas al norte y al sur.
Movimiento?
...
```

Y así sucesivamente. El juego está programado de la siguiente forma:

```
(put 'entrada 'descripcion "Te encuentras en la entrada de una cueva.
Hay un túnel hacia el este")
(put 'entrada 'este (lambda () (visit 'cueva)))
(put 'cueva 'descripcion "Te encuentras en una cueva amplia. Las
paredes brillan. Hay túneles al este y al sur. Hay una varita mágica
en el suelo")
(put 'cueva 'este (lambda () (visit 'camara-tortura)))
(put 'cueva 'sur (lambda () (visit 'biblioteca)))
(put 'varita 'coger (lambda () (visit 'sala-oro)))
(put 'varita 'agitar (lambda () (visit 'biblioteca)))
...
```

Tenemos los siguientes procedimientos (suponemos que `read-line` es una primitiva que devuelve la petición que el jugador escribe en forma de lista de identificadores):

```
(define (aventura)
      (visit 'entrada))

(define (visit habitacion)
      (newline)
      (display (get habitación 'descripcion))
      (newline)
      (display "Movimiento? ")
      (actua habitacion (read-line)))
```

Implementa el procedimiento `actua` que reciba la habitación donde se encuentra el jugador y una petición como argumento.

Se considera que todo lo que el jugador teclea es correcto.

Pregunta 6 (10 puntos)

Supongamos las siguientes expresiones en Scheme:

```
(define (g x)
   (let ((z 8))
      (lambda (y)
         (set! z (+ x y))
         (set! x (+ z y))
         (+ x y z))))
(define h (g 4))
(h 6)
```

a) (2 puntos) ¿Qué valor devolverá la última expresión?
b) (5 puntos) Dibuja y explica el diagrama de entornos creado al ejecutar las expresiones.
c) (3 puntos) ¿Tiene la función `g` estado local? ¿Tiene la función `h` estado local? ¿Qué variables contienen el estado local?