Lenguajes y Paradigmas de Programación
Curso 2007-2008
Examen de la Convocatoria de Junio

Normas importantes

- La puntuación total del examen es de 100 puntos sobre los que se valora la nota de la asignatura.
- Para sumar los puntos de las prácticas **es necesario obtener un mínimo de 40 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 estarán disponibles en la web de la asignatura el próximo día 1 de Julio

Pregunta 1 (16 puntos)

Escribe el procedimiento `alterna-f` que tome dos procedimientos como argumento y devuelva un procedimiento que los aplique alternativamente a los elementos de una lista.

```
((alterna-f square (lambda(x)(+ 1 x)) '(5 6 7 8))
(25 7 49 9)
(define updown (alterna-f (lambda (x) (+ x 100))
                          (lambda (x) (- x 100))))
(updown '(0 1 -1 2 -2 3 -3))
(100 -99 99 -98 98 -97 97)
```

Pregunta 2 (16 puntos)

Escribe el procedimiento `diff-pt` que tome como argumentos dos listas de listas (pseudoárboles) con la misma estructura, pero posiblemente con diferentes elementos, y devuelva una lista de parejas que contenga los diferentes elementos. Ejemplo:

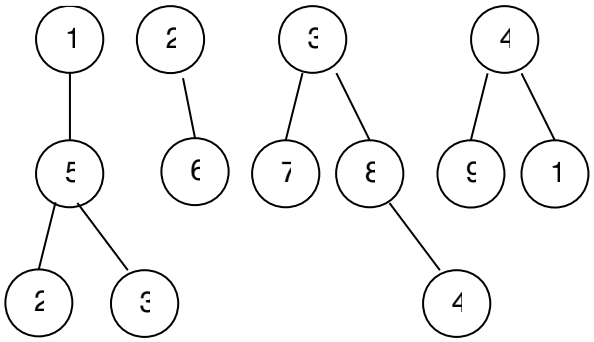
```
(diff-pt `(a (b ((c)) d e) f) '(1 (b ((2)) 3 4) f))
((a . 1) (c . 2) (d . 3) (e . 4))
(diff-pt '() '())
()
(diff-pt '((a b) c) '((a b) c))
()
```

Pregunta 3 (16 puntos)

Escribe el procedimiento `anchura-forest` que devuelva la máxima anchura de un bosque (lista de árboles). La anchura de un bosque en un nivel determinado se define como la suma del número de nodos que hay en esa nivel en todos los árboles del bosque. Por ejemplo, en la siguiente figura, la anchura del bosque en el nivel 1 sería 4, en el nivel 2 sería 6 y en el nivel 3 sería 3.

Ejemplo:

bosque:

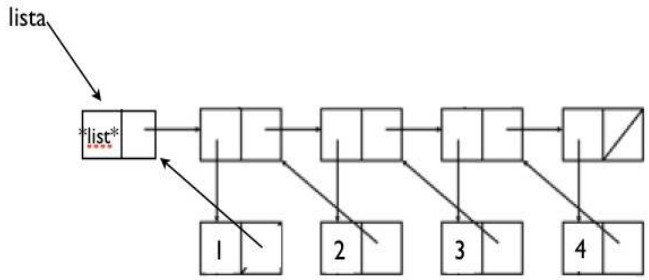


```
(anchura-forest bosque)
6
```

La anchura del bosque es 6 por ser la mayor anchura de todos los niveles.

Pregunta 4 (20 puntos)

Vamos a construir un nuevo tipo de dato, llamado *lista doblemente enlazada* o `dlist`. Esta estructura de datos tiene la propiedad de que, en cada momento, se puede acceder tanto al siguiente elemento como al anterior. Para implementarlo, definimos los elementos de esta lista como parejas cuya parte izquierda contiene el dato propiamente dicho y la parte derecha el enlace al elemento anterior. La lista va precedida de una pareja que hace el papel de cabecera y que es referenciada por las variables que apuntan a ella. En la figura puedes ver esta implementación:



Tenemos las siguientes funciones de su barrera de abstracción:

```
(define empty-dlist '())
(define (empty-dlist? obj)
        (null? Obj))
(define (first? l)
   (equal? '*list* (car l)))
(define (value dlist)
   (if (empty-dlist? dlist)
        (error "Error dlist vacía")
        (caar dlist)))
(define (next dlist)
   (if (empty-dlist? dlist)
        (error "Error dlist vacía")
        (cdr dlist)))
(define (previous dlist)

   (cond
        ((first? dlist) dlist)

        ((empty? dlist) (error "Error dlist vacía"))
```

```
        (else (cdar dlist))))
```

Vamos a completar la barrera de abstracción.

a) (6 puntos) Define el procedimiento `(make-dlist)` que construya una lista vacía y el procedimiento `(add-dilst dlist x)` que añada un primer elemento `x` en la primera posición de una lista vacía. La variable `dlist` referencia la cabecera de la lista.

a) (10 puntos) Define el procedimiento `(insert-dlist pos-dlist x)` que inserte un elemento `x` en la posición de la lista definida por `pos-dlist`. Esta posición puede ser la posición inicial (`pos-dlist` apuntando a la cabecera) o cualquier posición intermedia resultante de llamadas a `next`. !Cuidado con el caso especial del final de la lista¡

b) (4 puntos) Define el procedimiento `(list-to-dlist lista)` que construya una `dlist` con los elementos de la `lista` pasada como argumento.

Pregunta 5 (16 puntos)

Supongamos que representamos las fechas como un tipo de dato con los campos `día`, `mes` y `año`, todos ellos enteros. Queremos definir una función `fecha-to-string` a la que le pasemos un dato de tipo fecha y un símbolo identificando el formato de conversion de esa fecha. La función nos tiene que devolver una cadena con el formato escogido.

Ejemplo:

```
(define fecha (make-fecha 10 11 2007)) ; 10 de noviembre de 2007
(fecha-to-string fecha 'guiones-2-digitos) -> "10-11-07"
(fecha-to-string fecha 'guiones-alfa) -> "10-nov-2007"
(fecha-to-string fecha 'barras-4-digitos) -> "10/11/2007"
```

Queremos implementar la función `fecha-to-string` de forma que podamos añadir nuevos formatos **sin tener que reescribir su código**.

a) (8 puntos) Implementa la función `fecha-to-string` de forma que sea possible añadir nuevos formatos de fecha.

b) (8 puntos) Define una función para añadir los formatos de fecha, y escribe un ejemplo completo de funcionamiento, por ejemplo el correspondiente al formato `'guiones-alfa` del ejemplo anterior (recuerda que la función de Scheme para concatenar cadenas es `string-append`)

Pregunta 6 (16 puntos)

Supongamos las siguientes expresiones en Scheme:

```
(define y 5)
(define x 8)
(let ((x (+ y 3))
      (f (lambda (x)
           (+ x y)))
      (y 10))
  (f x))
```

Dibuja **(10 puntos)** y explica **(6 puntos)** el diagrama de entornos creado al ejecutar las expresiones.