Nombre:_____ Grupo:_____

Lenguajes y Paradigmas de Programación
Curso 2012-2013
Primer parcial - Turno de tarde

Normas importantes

- **La puntuación total del examen es de 10 puntos.**
- **Se debe contestar cada pregunta en las hojas que entregamos. Utiliza las últimas hojas para hacer pruebas. No olvides poner el nombre.**
- **La duración del examen es de 2 horas.**

Ejercicio 1 (1,5 puntos)

a) (0,75 puntos) Realiza la evaluación de las siguientes expresiones, utilizando el *orden normal* e indicando al final el resultado:

```
(define (f a b)
   (g (+ a b) a))
(define (g x y)
   (+ (* x y) y))
(f (- 3 2) (g 3 2))
```

b) (0,75 puntos) Indica qué devuelven las siguientes expresiones:

```
(define (misterio lista n)
  (map (lambda (x) (+ n x)) lista))
(misterio '(3 6 8) 4) → _____________

((lambda (a f b) (f a b)) 5 + 7) → _____________

((lambda (arg) (arg)) (lambda () 5)) → _____________
```

Ejercicio 2 (1,5 puntos)

Escribe la función `(cuenta-preds pred1 pred2 lista)` que reciba una `lista` y dos predicados y devuelva una pareja donde su parte izquierda contiene el número de elementos de la lista que cumplen el `pred1` y la parte derecha contiene los que cumplen el `pred2`.

```
(cuenta-preds (lambda (x) (> x 5)) (lambda (x) (odd? x)) '(1 2 3 4 5 6 7 8 9))
→ (4.5)
```

Ejercicio 3 (1,5 puntos)

Define la función `(mas-frecuentes lista1 lista2 n)` que reciba dos listas y un número. Devolverá una nueva lista que contenga aquellos elementos de la segunda lista que aparecen más de n veces en la primera. Puedes definir las funciones auxiliares que necesites.

Ejemplo:

```
(mas-frecuentes '(a b c d d 1 3 3 a b c c d d) '(a d b c) 3) → '(d c)
(mas-frecuentes '(a b c d d 1 3 3 a b c c d d) '(a d b c) 5) → '()
```

Ejercicio 4 (1,5 puntos)

Define la función `(genera-parejas-stream)` que devuelva una lista infinita de parejas, donde la parte izquierda de la pareja contiene un número entero n empezando en 1 y la parte derecha contiene un número desde 1 hasta `n.`

```
(genera-parejas-stream) →
((1.1)(2.1)(2.2)(3.1)(3.2)(3.3)(4.1)(4.2)(4.3)(4.4)(5.1)...)
```

Ejercicio 5 (2 puntos)

a) (1 punto) Supongamos el siguiente código en Scheme

```
(define (g a b)
   (* a b 10))
(define (f x y)
   (+ x y 6))
(f (g 2 3) (f 4 5))
```

Junto a cada entorno escribe un número indicando en qué orden se ha creado. Explica la evaluación de las expresiones haciendo referencia a esos números. ¿Cuál es el resultado? ¿Cuántos ámbitos se crean? ¿En qué orden? ¿Se crea alguna clausura?

b) (1 punto) Supongamos el siguiente código en Scheme

```
(define y 10)
(define (g f x)
   (let ((y 30))
      (+ y (f x))))
(g (lambda (z) (+ z y)) 20)
```

Junto a cada entorno escribe un número indicando en qué orden se ha creado. Explica la evaluación de las expresiones haciendo referencia a esos números. ¿Cuál es el resultado? ¿Cuántos ámbitos se crean? ¿En qué orden? ¿Se crea alguna clausura?

Ejercicio 6 (2 puntos)

a) (1 punto) Describe 5 de los hechos más importantes de la historia de la computación entre 1930 y 1950. Escribe un párrafo sobre cada uno de ellos explicando por ejemplo, fechas, científicos, características, conceptos, etc.

b) (0,2 puntos) En Scheme, un ejemplo de abstracción es (rodea las opciones que sean correctas):

a) La posibilidad de definir funciones con la forma especial define
b) El uso de expresiones bien delimitadas por paréntesis
c) La posibilidad de usar guiones en los nombres de los identificadores
d) La definición del tipo de datos lista, y sus funciones list, list?, length, null?, list-ref, etc.
e) El uso de la forma especial lambda para crear funciones anónimas

c) (0,2 puntos) El nombre de Lisp viene de (rodea la correcta):

a) Lambda Processing
b) Lots of Insipid and Stupid Parenthesis
c) List Instance Sentence Programs
d) List Processing

d) (0,2 puntos) Un *side effect* se produce en (rodea las opciones que sean correctas):

a) Los lenguajes de programación mal diseñados como el BASIC
b) Los lenguajes von Neumann basados en sentencias y asignación en memoria
c) Los lenguajes de programación funcionales
d) Cualquier lenguaje completo equivalente a una Máquina de Turing
e) Cualquier lenguaje imperativo

e) (0,2 puntos) Un *l-value* es (rodea la correcta):

a) Un puntero o referencia que se guarda en una variable
b) Un valor constante e inmutable que podemos usar en programación funcional
c) Una expresión en la que puede guardarse un valor o una referencia
d) Una expresión que produce el valor utilizado en una asignación

f) (0,2 puntos)

Dadas las siguientes definiciones de delay y force:

```
(define (new-delay expr)  (lambda() expr))
(define (new-force promise) (promise))
```

Explica si su comportamiento sería igual a las formas especiales delay y force que conoces.