Nombre:_____ **Grupo:**_____

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

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

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

```
(map (lambda (x y) (list x y)) '(a b c) '(j k l)) → _____________

((lambda (a f b) (f a b)) 2 - 4) → _________________

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

Ejercicio 2 (1,5 puntos)

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

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

Ejercicio 3 (1,5 puntos)

Define una función recursiva llamada `ocurrencias-elementos` que tome como argumentos dos listas y devuelva una lista de parejas, en donde cada pareja contiene en su parte izquierda un elemento de la segunda lista y en su parte derecha el número de veces que aparece dicho elemento en la primera lista. Puedes definir las funciones auxiliares que necesites.

Ejemplo :

```
(ocurrencias-elementos '(1 3 6 2 4 7 3 9 7) '(5 2 3)) →
((5 . 0) (2 . 1) (3 . 2))
```

Ejercicio 4 (1,5 puntos)

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

```
(genera-parejas-stream) →
((1.1)(1.2)(2.2)(1.3)(2.3)(3.3)(1.4)(2.4)(3.4)(4.4)(1.5)...)
```

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))
```

Dibuja en un diagrama los entornos que se generan. 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? ¿Se crea alguna clausura?

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

```
(define x 3)
(define y 8)
(define (g h x)
   (h (+ x y)))
(let ((x 13)
        (y 18))
     (g  (lambda (x) (+ x y)) (+ x 3)))
```

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? ¿Se crea alguna clausura?

Ejercicio 6 (2 puntos)

a) (1 punto) Describe 5 de los lenguajes de programación más importantes desarrollados *hasta el año 2000*. Escribe un párrafo sobre cada uno de ellos explicando, por ejemplo, fechas, investigadores que lo han desarrollado, características, paradigma en el que se inscriben, tipo de ejecución (compilado o interpretado), etc.

b) (0,2 puntos) El uso de la abstracción en los programas permite (rodea las opciones que sean correctas)

1) Facilitar la visualización del funcionamiento del programa y del sistema software
2) Implementar software que corre de una forma más eficiente y rápida
3) Encontrar más fácilmente los errores en un programa
4) Obtener software más modular en el que las partes pueden ser construidas, reemplazadas y probadas por separado

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

1) Lambda Processing
2) Lots of Insipid and Stupid Parenthesis
3) List Instance Sentence Programs
4) List Processing

d) (0,2 puntos) ¿Cuál de las siguientes afirmaciones son ciertas (puede haber más de una)?

a) En Pascal es posible pasar funciones como argumentos de otras funciones, pero hay que especificar el tipo de la función, qué tipos recibe y devuelve.
b) En Pascal es posible pasar funciones como argumentos de otras funciones, y el tipo de la función se infiere en tiempo de compilación.
c) En C y C++ se pueden pasar como parámetros o asignar a variables punteros a funciones.
d) Las funciones son objetos de primera clase en C y C++.
e) En Objective-C las clausuras se denominan *bloques*

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) Explica qué diferencias existen entre `eval` y `apply`. Pon algún ejemplo.