Nombre:_____ **Grupo:**_____

Lenguajes y Paradigmas de Programación
Curso 2013-2014
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 (2 puntos)

a) (0,75 puntos) Explica la función de orden superior fold: qué parámetros acepta y qué valores devuelve. Escribe su implementación recursiva y pon un ejemplo de su funcionamiento.

b) (0,75 puntos) Explica las distintas definiciones y características de los lenguajes de programación.

c) (0,25 puntos) Indica el orden temporal de los lenguajes: Python, LISP, C, Go

d) (0,25 puntos) Indica el orden temporal de los siguientes hitos históricos relacionados con la historia de los computadores:

- Arquitectura Von Neumann
- Máquina de Turing
- Motor de diferencias de Babbage
- Calculadora de Pascal

Ejercicio 2 (2 puntos)

a) (0,5 puntos) Para cada una de las siguientes expresiones, da una definición de `f` que sea correcta:

```
((f 2))
```

```
(f (f 4))
```

b) (0,5 puntos) Rellena los huecos:

```
(map car (list (cons 1 2) (cons 3 4) (cons 5 6))) → _____________
```

```
(apply list (list (cons 1 2) (cons 4 5))) → _____________
```

c) (0,5 puntos) Rellena los huecos para obtener el resultado esperado (puedes utilizar `string-length`):

```
(fold ________  _____ '("x" "abc" "xyzzz" "jk")) ; num caracteres de la palabra
                                                 ; más larga
→ 5
```

d) (0,5 puntos) Dibuja el *Box&Pointer* de la siguiente expresión y explica si genera una lista o no:

```
(cons (cons 1 (cons 2 3)) (cons (list 1 2) (list 1 2)))
```

Ejercicio 3 (2 puntos)

a) (1 punto) Define la función recursiva `(anteriores lista x)` que reciba una lista y un elemento y devuelva una lista con los elementos anteriores de `x` en la lista. El elemento puede aparecer más de una vez.

Ejemplos:

```
(anteriores '(a b c d a c b c a) 'b) → (a c)
(anteriores '(b w b b c d a c b c a t b) 'b) → (w b c t)
```

b) (1 punto) Define la función recursiva `(total x lista-parejas)` que recibe un símbolo `x` y una lista de parejas con símbolos en su parte izquierda y números en su parte derecha. Debe devolver una pareja que contiene el símbolo `x` y la suma de todas las partes derechas en las que aparece `x`.

Ejemplo:

```
(total 'a '((a . 2) (b . 3) (c. 8) (a . 1) (a . 10))) → (a . 13)
```

Ejercicio 4 (2 puntos)

a) (0,75 puntos) Utilizando la función de orden superior que consideres más apropiada, define la función `(mayores-izq lista-parejas)` que reciba una lista de parejas de números y devuelva una lista con aquellas parejas cuya parte izquierda sea mayor que la parte derecha.

Ejemplo:

```
(mayores-izq '((2.2)(6.2)(4.5)(5.5)(7.1))) → ((6.2)(7.1))
```

b) (1,25 puntos) Define la función `(aplica-if lista-funcs lista-bools n)` que reciba dos listas con el mismo número de elementos, una de ellas contiene funciones unarias y la otra valores booleanos que indican si las funciones situadas en esas posiciones de la primera lista se deben usar o no, y un número. Esta función deberá aplicar las funciones de `lista-funcs` seleccionadas al número `n`. Puedes utilizar funciones auxiliares y/o de orden superior.

Ejemplo:

```
(aplica-if (list suma-1 mult-3 doble cuadrado) '(#f #t #f #t) 3)
→ 27  ; (mult-3 (cuadrado 3))
```

Ejercicio 5 (2 puntos)

Dados los siguientes fragmentos de código en Scheme, dibuja en un diagrama los ámbitos que se generan. Junto a cada ámbito escribe un número indicando en qué orden se ha creado. ¿Cuál es el resultado? ¿Cuántos ámbitos se crean? ¿Se crea alguna clausura?

a) (1 punto)

```
(define k 8)
(define (prueba2 x)
    (- x k))
(define a
    (let ((x 10))
        (prueba2 x))
```

b) (1 punto)

```
(define z 10)
(define (prueba x y)
   (+ x y z))
(define (prueba2 z)
   (lambda (x y)
      (prueba x y)))
(define g (prueba2 12))
(g 14 16)
```


Eval
Apply
λ