Nombre y Apellido: _____ Comisión: _____

PARADIGMAS DE PROGRAMACIÓN

Recuperatorio 3er Parcial: Programación Funcional

Fecha: 10/02/2023

INSTRUCCIONES:

- ✓ **Las soluciones de los Ejercicios 1, 2a y 2b deben entregarse en esta hoja.**
- ✓ **Resolver los Ejercicios 2c, 3 y 4 cada uno en una hoja separada.**
- ✓ **Poner nombre en TODAS las hojas que entregue.**
- ✓ **Numerar TODAS las hojas indicando en cada una el total (nro. hoja/total).**

Ejercicio 1 (25 puntos)

a) En el Cálculo Lambda, el término (Q P):

	Es un redex solamente si Q es una abstracción funcional, independientemente de la forma de P
	Es un redex sólo si Q y P son ambas abstracciones funcionales
	Nunca podría ser un redex
	Es un redex solamente cuando P es una aplicación funcional, independientemente de la forma de Q
	Es un redex independientemente de la forma de Q y P
	Ninguna de las anteriores es correcta

b) En el Cálculo Lambda:

	Un término que está en forma normal no puede ser reducido
	Un término que no contiene un redex puede ser reducido aplicando la regla Beta
	Al aplicar la regla Beta a un término que tiene más de un redex, el mismo siempre se reduce al mismo término independientemente de la estrategia utilizada (reducción por orden normal o reducción por orden aplicativo)
	Si un término tiene forma normal, siempre es posible hallarla independientemente de la estrategia de reducción utilizada (orden normal u orden aplicativo)
	Ninguna de las afirmaciones anteriores es correcta.

c) El término `λz.λf.(λx.x (f f))`

	Es una aplicación funcional
	Representa una función, definida sobre la variable z, que al ser aplicada a un argumento efectivo permite obtener una abstracción funcional
	Representa una función, definida sobre la variable z, que al ser aplicada a un argumento efectivo permite obtener una aplicación funcional
	Está en forma normal
	Ninguna de las afirmaciones anteriores es correcta.

d) Al evaluar las siguientes expresiones en scheme,

e1. `(append (cadddr '(a b c (d))) '(hola))`

e2. `(list (cadddr '(a b c (d))) 'hola)`

e3. `(cons (cadddr '(a b c (d))) '(hola))`

	Las tres producen el mismo resultado
	Las tres producen como resultado una lista
	e3 y e1 producen el mismo resultado
	e1 y e2 producen el mismo resultado
	Ninguna de las anteriores

e) El lenguaje Scheme:

	Siempre utiliza orden normal para evaluar las expresiones
	A veces utiliza orden normal para evaluar las expresiones, dependiendo del tipo de expresión que se evalúa
	No admite funciones como argumentos de funciones
	Sólo admite funciones de 1 argumento como argumento de otras funciones
	Al evaluar la función (lambda (z) (if (null? z) z (lambda (x) (z x)))), puede retornar una lista
	Ninguna de las afirmaciones anteriores es correcta.

Ejercicio 2 (15 puntos)

Considere el siguiente término del Cálculo Lambda:

$$(\lambda y.\lambda x.(\lambda z.(z\ y)\ x)\ \ (\lambda w.(w\ y)\ x))$$

a) (3 puntos) Marque en la expresión anterior, cuántos y cuáles (si hubiera) son los redex que contiene.

b) (2 puntos) Para cada variable, indicar el número de ocurrencias libres y ligadas de la misma, completando la siguiente tabla:

Variable	Ocurrencias Libres	Ocurrencias Ligadas

c) (10 puntos) Hallar la forma normal del término indicando claramente las reglas utilizadas en cada paso.

Ejercicio 3 (25 puntos)

Definir las siguientes funciones de orden superior:

a) (10 puntos)

Una función de orden superior de 2 argumentos, uno de los cuales es una función:

(**valorExtremo** ***criterio lista***)

criterio: es una función de 2 argumentos que retorna **#t** o **#f**.

lista: es una lista homogénea, es decir todos sus elementos son de un mismo tipo (número, carácter, string, lista, pares, etc)

La función **valorExtremo** retorna el elemento de la lista que hace que ***criterio*** retorne **#t** cuando es evaluada con este elemento como primer argumento y como segundo argumento, cualquiera de los otros elementos de la lista. Por ejemplo, si la lista contiene números y la función ***criterio*** es la función mayor (**>**), la función **valorExtremo** retornará el mayor elemento de ***lista***, si la función ***criterio*** es la función menor (**<**), la función **valorExtremo** retornará el menor valor de ***lista***.

Ejemplos:

```
> (valorExtremo < '(2 3 1 0 9 7 11 10))
0     ; en este caso la función criterio es <

> (valorExtremo > '(2 3 1 0 9 7 11 10))
11    ; en este caso la función criterio es >

> (extremo char>=? '(#\h #\w #\p #\s))
#\w   ; en este caso la función criterio es char>=?

> (extremo (lambda (l1 l2) (<= (length l1) (length l2)))
           '((1 2) (1 2 3) (1) (1 2 3 4)))
'(1)  ; en este caso la función criterio es
          (lambda (l1 l2) (<= (length l1) (length l2)))
```

b) (15 puntos)

Una función de orden superior de 1 argumento, que retorna una función de 1 argumento:

(**ordenGenerico** ***criterio***)

criterio: es una función de 2 argumentos como la descripta en el apartado a).

La función retornada por **ordenGenerico** será de 1 argumento, el cual deberá ser una lista homogénea. Cuando esta función se aplique a una lista, retornará una lista con los elementos de la lista argumento, pero ordenados de acuerdo a ***criterio***.

Por ejemplo:

```
> (define ordenaMayorMenor (ordenGenerico >))
> (ordenMayorMenor '(2 1 3 8 6 7))
'(8 7 6 3 2 1)

> ((ordenGenerico char<=?) '(#\h #\w #\p #\s))
'(#\h #\p #\s #\w)
```

Ayuda: para la definición de la función **ordenGenerico** utilice la función definida en el apartado **a)**. También puede ser necesario definir funciones auxiliares.

Ejercicio 4 (35 puntos)

Se cuenta con información sobre los partidos jugados en el Mundial de Rugby Sub 20 en 2019. La información está representada por listas en Scheme con la estructura que se indica a continuación.

Información de jugadores, por ejemplo:

```
(define lista-jugadores
    '((hocquet francia) (zegueur francia) (hamonou francia) (delord francia)
      (perez argentina) (gonzalez argentina) (pedemonte argentina) …))
```

Donde cada sublista (*jugador equipo*) representa la relación entre un jugador y el equipo al que pertenece.

Información de partidos, por ejemplo:

```
(define lista-partidos
    '((p1 (8 6 19) (sudafrica 48) (georgia 20))
      (p2 (12 6 19) (sudafrica 25) (nueva-zelanda 17))
      (p3 (4 6 19) (sudafrica 43) (escocia 19))
      (p4 (12 6 19) (francia 26) (argentina 47))
      (p5 (8 6 19) (argentina 41) (fiji 14))
      (p6 (4 6 19) (argentina 25) (gales 30))
      (p7 (8 6 19) (francia 32) (gales 13)))
```

Donde cada sublista (*nro-partido* (*día mes año*) (*equipo1 ptos1*) (*equipo2 ptos2*)) representa la relación entre los equipos que jugaron un partido en una fecha determinada y los puntos obtenidos por cada uno de ellos.

Además, se dispone de una lista que indica los partidos jugados por cada jugador.

```
(define partidos-jugados
    '((p4 gonzalez) (p4 pedemonte) (p4 perez) (p4 hocquet) (p5 gonzalez)
      (p5 pedemonte) (p6 perez) (p7 hocquet) (p7 hamonou) (p7 delord) …))
```

Donde cada lista (*nro-partido jugador*) representa un partido que jugó un jugador.

En base a las listas anteriores se define la información de un torneo como una lista de 3 elementos con:

```
(define info-torneo (list lista-jugadores lista-partidos partidos-jugados))
```

Defina las siguientes funciones en Scheme:

a) (partidos-ganados *equipo lista-partidos***)**

Que retorna una lista con los partidos ganados por el equipo dado como primer argumento. **En la lista resultante los partidos deben estar ordenados ascendentemente según la fecha (*día mes año*) en la que fueron jugados**, para lo cual puede asumir que ningún equipo participa en dos partidos en la misma fecha.

Considere los siguientes ejemplos:

```
> (partidos-ganados 'sudafrica lista-partidos)
'(p3 p1 p2)
> (partidos-ganados 'argentina lista-partidos)
'(p5 p4)
```

b) (jugadores-partido *equipo nro-partido info-torneo***)**

Que retorna una lista con los jugadores que jugaron el partido *nro-partido* para el *equipo* indicado. Por ejemplo:

```
> (jugadores-partido 'argentina 'p4 info-torneo)
'(gonzalez pedemonte perez)
```

c) (partidos-ganados-jugador *jugador info-torneo***)**

Que devuelve una lista con los partidos ganados por el *jugador* indicado. El orden de los partidos es indistinto. Ejemplos:

```
> (partido-ganados-por 'gonzalez info-torneo)
'(p4 p5)
> (partido-ganados-por 'perez info-torneo)
'(p4)
```

d) (jugador-afortunado *jugador info-torneo***)**

Que retorna verdadero si el jugador indicado ha ganado todos los partidos que ha jugado, caso contrario retorna falso. Por ejemplo:

```
> (jugador-afortunado 'pedemonte info-torneo)
#t
> (jugador-afortunado 'delord info-torneo)
#t
> (jugador-afortunado 'perez info-torneo)
#f
```