Nombre y Apellido: _____ Comisión: _____

PARADIGMAS DE PROGRAMACIÓN

3er Parcial: Programación Funcional

Fecha: 12/12/2022

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 (λz.Q P)

	Es un redex solamente si Q es una abstracción funcional
	Es un redex solamente si Q y P son abstracciones funcionales
	Podría ser un redex solamente cuando P es una abstracción funcional
	Podría ser un redex solamente cuando P es una aplicación funcional
	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 puede ser reducido
	Un término que tiene al menos 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 λx.λf.(x (f f))

	No se puede expresar en Scheme ya que no posee variables libres
	Presenta a x como variable libre y ligada
	En Scheme definiría una función sin nombre de orden superior
	Posee un redex
	Ninguna de las afirmaciones anteriores es correcta.

d) Al evaluar las siguientes expresiones en Scheme,

e1. `(append (cadddr '(1 2 3 (4))) '(hola))`

e2. `(cons (cadddr '(1 2 3 (4))) '(hola))`

e3. `(cons (cadddr '(1 2 3 (4))) 'hola)`

	Las tres producen el mismo resultado
	Las tres producen como resultado una lista
	e3 da como resultado un par y las otras dos expresiones devuelven listas

	e1 y e2 producen el mismo resultado
	Ninguna de las anteriores

e) En el Cálculo Lambda

	La regla Beta puede expresarse como **(λx.N M) = [N/x] M**
	La aplicación de la regla beta en la expresión **[λz.(z y) / x] λx.(x y)** da como resultado la expresión: **[λz.(z y) / y] [y / x] (x y)**
	La aplicación de la regla beta en la expresión **[λx.(x y) / x] λx.(x y)** da como resultado la expresión: **λx.[λx.(x y) / x] (x y)**
	La aplicación de la regla beta en la expresión **[λz.(z y) / x] λx.(x y)** da como resultado la expresión: **λx.(x y)**
	Ninguna de las afirmaciones anteriores es correcta.

Ejercicio 2 (15 puntos)

Considere el siguiente término del Cálculo Lambda:

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

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

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

Variable	Ocurrencias Libres	Ocurrencias Ligadas

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

Ejercicio 3 (25 puntos)

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

(**reduceIz** ***f***)

Donde ***f*** es una función de 2 argumentos.

El resultado de evaluar (**reduceIz** ***f***) es una función de 1 argumento, el cual debe ser una lista.

Esta función reduce la lista a 1 valor mediante la aplicación de ***f*** a los elementos de la lista. La primera vez se aplica ***f*** a los primeros 2 elementos de la lista. A continuación, ***f*** se aplicará

tomando como primer argumento el resultado anterior y como segundo argumento el próximo elemento de la lista. Se repite la aplicación de ***f*** tomando como primer argumento el valor acumulado hasta ese punto y como segundo argumento el próximo elemento de la lista, hasta finalizar. En este punto, la lista tiene 1 sólo elemento y se retorna ese elemento. Puede asumir además que la lista nunca estará vacía.

Atención: la aplicación de **reduceIz** retorna una función de 1 argumento

Ayuda: en el recorrido recursivo de la lista, puede ir reemplazando los 2 primeros elementos por el resultado de la aplicación de ***f*** a los mismos.

Por ejemplo:

```
> ((reduceIz +) '(1 2 3 4 10))
20
> (max 1 2)
2
> ((reduceIz max) '(3 5 1 8 6 7))
8
> ((reduceIz append) '((3 5) (1 8 6) (7 15 10)))
'(3 5 1 8 6 7 15 10)
```

Ejercicio 4 (35 puntos)

Cansado de utilizar su ex-planilla de cálculo favorita, un profesor desea desarrollar un método alternativo para calcular los resultados finales de sus alumnos en un examen final.

Utilizando Scheme, decide representar un examen como una lista de ejercicios. A su vez, propone representar cada ejercicio como una lista consistente de un código, un tema y una nota máxima. Por ejemplo, un examen final de Paradigmas podría estar representado por la siguiente lista:

```
(define examenParadigmas
  '((ej1 logica 6) (ej2 logica 24) (ej3 objetos 6)
    (ej4 objetos 34) (ej5 funcional 6) (ej6 funcional 24)) )
```

Por otra parte, los resultados de un examen para un alumno dado se representarán como listas con la siguiente estructura:

'(*nombreAlumno* (*ejercicio*$_1$ *nota*$_1$) (*ejercicio*$_2$ *nota*$_2$) ...)

Donde *nombreAlumno* será una lista conteniendo los nombres y finalizando con el apellido del alumno (al final de la lista).
Además, cuando la nota de un ejercicio es 0 (cero) no se incluye la misma en el listado de notas. Por ejemplo, los resultados del alumno Ariel J. Prieto serán representados como:

```
(define resultadosAriel
 '((ariel j prieto) (ej1 6) (ej2 24) (ej3 6)
                    (ej4 30) (ej5 6) (ej6 24)))
```

Considere además la siguiente lista, que incluye los resultados de tres alumnos:

```
(define listaResultados
 '( ((nicolas lopez) (ej1 2) (ej2 10) (ej4 8) (ej6 5))
    ((juan l perez) (ej2 4) (ej4 12) (ej5 2) (ej6 3))
    ((ariel j prieto) (ej1 6) (ej2 24) (ej3 6)
                      (ej4 30) (ej5 6) (ej6 24)) ) )
```

Habiendo definido la representación a utilizar, se le solicita que desarrolle en Scheme las siguientes funciones:

a) (maximaNota *examen*)

Calcula la nota total máxima que un alumno puede obtener en el examen pasado como argumento.

Ejemplo:

```
> (maximaNota examenParadigmas)
100
```

b) (notaTotalAlumno *examen resultados*)

Calcula la nota total de un alumno en base a los resultados del mismo. Si la nota obtenida es mayor a la nota máxima del examen debe retornar el símbolo 'error.

Ejemplo:

```
> (notaTotalAlumno examenParadigmas resultadosAriel)
96
```

c) (armarPlanilla *examen listaResultados*)

Permite obtener una planilla con los resultados por alumno para un examen dado. Para cada alumno se indicará su apellido, su nota total y su condición. La condición será "aprobado" si la nota final es mayor o igual al 60% de la nota máxima del examen. Caso contrario será "no-aprobado".

Ejemplo:

```
> (armarPlanilla examenParadigmas listaResultados )
'((lopez 25 no-aprobado) (perez 21 no-aprobado) (prieto 96
aprobado))
```

d) (filtraPlanilla *criterio*)

Permite obtener una función que recibe como argumentos una planilla (como la devuelta por la función del ítem anterior) y genera una lista con los apellidos de los alumnos cuyo resultado cumple con el argumento criterio. **filtraPlanilla** al ser aplicada a un ***criterio*** retorna una función de 1 argumento, el cual debe ser una lista.

Ejemplos:

```
> (filtraPlanilla 'no-aprobado)
#<procedure>
> ((filtraPlanilla 'no-aprobado) '((lopez 25 no-aprobado) (perez
21 no-aprobado) (prieto 96 aprobado)))
'(lopez perez)
> ((filtraPlanilla 'aprobado) '((lopez 25 no-aprobado) (perez 21
no-aprobado) (prieto 96 aprobado)))
'(prieto)
```