Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación 2017-1
Tarea 3

Karla Ramírez Pulido
José Ricardo Rodríguez Abreu
Manuel Soto Romero

Fecha de inicio: Martes 21 de Septiembre.
Fecha de entrega: Martes 11 de Octubre.

1 Lineamientos de la tarea

1. Esta tarea puede ser entregada en equipos de máximo 3 integrantes.
2. Se recibirá a mano con letra legible o de manera digital escrito en LaTeX en cuyo caso se deberá enviar el archivo .tex y .pdf al correo `ricardo_rodab@ciencias.unam.mx` con el asunto `[lenguajes]Tarea03`.
3. En caso de que alguna respuesta involucre escribir algún pequeño programa, éste será enviado a más tardar el día de la fecha de entrega al correo `ricardo_rodab@ciencias.unam.mx` con el asunto `[lenguajes]Tarea03-Codigos` y cada archivo será nombrado con el número de pregunta que le corresponda.
4. La fecha de entrega es única y no habrá prorroga.
5. Se deberá anexar al final de la tarea la bibliografía consultada. Cualquier tarea con material consultado y no citado propiamente se le considerará plagio y será promediado con cero.

2 Responda correctamente cada inciso

2.1 (2 ptos) En clase hemos visto el lenguaje FAE, que es un lenguaje con expresiones aritméticas, funciones y aplicaciones de funciones. ¿Es FAE un lenguaje Turing-Completo? Debes proveer una respuesta breve e inambigua, seguida de una justificación más extensa de tu respuesta. Hint: Investiguen sobre el combinador Y.

2.2 (2 ptos) ¿Java es glotón o perezoso? Escribe un programa para determinar la respuesta a esta pregunta. El mismo programa, ejecutado en cada uno de los dos regímenes, debe producir resultados distintos. Puedes usar todas las características de Java que gustes, pero debes mantener el programa relativamente corto: **penalizaremos cualquier programa que consideremos excesivamente largo o confuso** (Hint: es posible resolver este problema con un programa de unas cuantas docenas de lineas). Debes anexar tanto el código fuente de tu programa (Punto 1.3 de los linemientos de la tarea) así como una respuesta a la pregunta de si Java es glotón o perezoso, y una explicación de porque su programa determina esto. Es decir, deben proveer una respuesta breve e inambigua (p.ej. "Java es perezoso") seguida de una descripción del resultado que obtendrías bajo cada régimen, junto con una breve explicación de por que ese régimen generaría tal resultado.

2.3 (2 ptos) Ningún lenguaje perezoso en la historia ha tenido operaciones de estado (tales como la mutación de valores en cajas o asignación de valores a variables) ¿Por qué no? La mejor respuesta a esta pregunta incluiría dos cosas: un pequeño programa (que asume la evaluación perezosa) el cual usara estado y una breve explicación de cual es el problema que ilustra la ejecución de dicho programa. Por favor usa la noción original (sin cache) de perezosos sin cambio alguno. Si presentas un ejemplo lo suficientemente ilustrativo (el cual no necesita ser muy largo), tu explicación sera muy pequeña.

2.4 (2 ptos) Invesigue las funciones lambda en Ruby y conteste las siguientes preguntas:

- ¿Qué tipo de evaluación utilizan las funciones lambda en Ruby? Escribe y anexa un pequeño programa donde justifique su razonamiento.
- Con base al Cuadro 1.2 que se encuentra en el anexo:
  - ¿El resultado de ejecutar el programa cambia respecto a otros lenguajes (como Java o C++) por el tipo de evaluación que reciben las funciones lambda? Justifica tu respuesta.
  - ¿Cuál es la función del método `call()` en el programa?
  - Ejecuta en tu computadora el programa del Cuadro 1.2 y explica línea a línea la evaluación del mismo ignorando los comentarios.

2.5 (3 ptos) Lea cuidadosamente y responda los incisos del 2.5.1 al 2.5.5.

Se escribe el programa del Cuadro 1.1 que se encuentra en el anexo en un archivo llamado `Ejemplo.rb`, en el lenguaje de programación Ruby. En este programa pasa lo siguiente:

a) El programa en `Ejemplo.rb` utiliza una biblioteca (gema en Ruby) llamada `'lazy'`. Lo que hace `'lazy'` es proporcionar las funciones `promise()` y `demand()`.

- `promise()`- Lo que hace es forzar la evaluación perezosa en lugar de la evaluación glotona.
- `demand()` - Evalúa todo lo que reciba de parámetro de forma glotona (incluyendo aquellos objetos que haya sido etiquetados "de evaluación perezosa" o tipo `Lazy`).

b) Las asignaciones de `promise()` en este programa de Ruby no son un punto estricto: Nos regresan un objeto de tipo `Lazy` que será evaluado bajo algunas circunstancias.

2.5.1 Menciona al menos 5 circunstancias en las que **tu creas** que los objetos de tipo `Lazy` se evalúan en un programa de Ruby que use `promise()` y `demand()`.

2.5.2 Señala los puntos estrictos del programa (script) `Ejemplo.rb`.

2.5.3 Escribe el programa `Ejemplo.rb` en tu computadora y ejecútalo ¿Cuál fue tu salida? Explíca la salida y la evaluación parámetro a parámetro. (Hint: Puedes ir colocando funciones "p" para imprimir variables entre cada línea)

2.5.4 La variable `'e'` está definida de la siguiente manera: `'e = d*2'` pero ni `'e'` ni la variable `'d'` son evaluadas.

- ¿Por qué no son evaluadas todas las variables aún al usar su valor en la asignación de otras variables?
- ¿Afecta esto el valor de la variable `'e'` si es que la variable `'d'` mutara antes de que `'e'` fuera evaluada? Justifica tu respuesta con un ejemplo y explícalo minuciosamente.

2.5.5 ¿Las funciones de la gema Lazy son equivalentes a las funciones lambda de Ruby? Justifique su respuesta.

3 Anexo

```ruby
# -----------------------------------------------------------------------
# Ejemplo.rb
# versión 1.0
# Copyright (C) 2016  José Ricardo Rodríguez Abreu.
# Facultad de Ciencias,
# Universidad Nacional Autónoma de México, Mexico.
#
# Este programa es software libre; se puede redistribuir
# y/o modificar en los términos establecidos por la
# Licencia Pública General de GNU tal como fue publicada
# por la Free Software Foundation en la versión 2 o
# superior.
#
# Este programa es distribuido con la esperanza de que
# resulte de utilidad, pero SIN GARANTÍA ALGUNA; de hecho
# sin la garantía implícita de COMERCIALIZACIÓN o
# ADECUACIÓN PARA PROPÓSITOS PARTICULARES. Véase la
# Licencia Pública General de GNU para mayores detalles.
#
# Con este programa se debe haber recibido una copia de la
# Licencia Pública General de GNU, de no ser así, visite el
# siguiente URL:
# http://www.gnu.org/licenses/gpl.html
# o escriba a la Free Software Foundation Inc.,
# 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
# -----------------------------------------------------------------------

require 'lazy'

# Función que realmente no hace nada interesante.
def fun(a, b, random)
  c = promise { b*-1 }
  d = promise { a+b+c }
  e = promise { d*2 }
  f = promise { a*random.rand(100) }
  g = promise { (b-1)*random.rand(100) }
  if (g-f) == 1
    return 1
  else
    # Este return sí es válido en Ruby:
    # regresa un arreglo con todos los parametros que le reciba en return.
    return demand(a),b,c*2,d,e,f,g
  end
end

a = promise { 5*2 }
b = promise { a+2 }
r = promise { Random.new }

# Usamos la función "p foo" porque imprime "foo.inspect" en lugar de "foo.to_s".
p fun(a, b, r)
```

Cuardo 1.1: `Ejemplo.rb`

```ruby
# ------------------------------------------------------------------------
# Lambdas.rb
# versión 1.0
# Copyright (C) 2016  José Ricardo Rodríguez Abreu.
# Facultad de Ciencias,
# Universidad Nacional Autónoma de México, Mexico.
#
# Este programa es software libre; se puede redistribuir
# y/o modificar en los términos establecidos por la
# Licencia Pública General de GNU tal como fue publicada
# por la Free Software Foundation en la versión 2 o
# superior.
#
# Este programa es distribuido con la esperanza de que
# resulte de utilidad, pero SIN GARANTÍA ALGUNA; de hecho
# sin la garantía implícita de COMERCIALIZACIÓN o
# ADECUACIÓN PARA PROPÓSITOS PARTICULARES. Véase la
# Licencia Pública General de GNU para mayores detalles.
#
# Con este programa se debe haber recibido una copia de la
# Licencia Pública General de GNU, de no ser así, visite el
# siguiente URL:
# http://www.gnu.org/licenses/gpl.html
# o escriba a la Free Software Foundation Inc.,
# 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
# ------------------------------------------------------------------------

var_a = nil

lambda_0 = lambda { if var_a == nil
                      var_a = 5 + 5
                    else
                      var_a += var_a
                    end }

lambda_1 = lambda { lambda_0.call() + lambda_0.call() }
lambda_2 = lambda { lambda { lambda_1.call() + lambda_1.call() } }

# Es una función que nos genera un  ciclo infinito.
lambda_3 = lambda { while true do
                      p var_a
                      var_a += var_a
                    end }

var_a = lambda_2.call()

if var_a == 10
  puts "Ciclo infinito! Oh no!"
  lambda_3.call()
elsif var_a == 150
  puts("Ups!")
  lambda_3.call()
else
  puts "¿Qué pasó?"
end
```

Cuardo 1.2: `Lambdas.rb`

4 Enlaces de ayuda

1. μ-recursividad https://en.wikipedia.org/wiki/%CE%9C-recursive_function

2. “¿Qué son las lambdas en Ruby?” http://culttt.com/2015/05/13/what-are-lambdas-in-ruby/

3. “Lazy.rb documentation” http://moonbase.rydia.net/software/lazy.rb/0.9.5/doc/

4. “Lazy.rb examples” https://www.ruby-forum.com/topic/55173

5 Formato bibliografía

El formato que debe llevar la bibliografía es el siguiente:
Nombre del autor empezando por apellidos, “Nombre del libro”, Editorial, Edición, País, No. de páginas del libro.
ó
Nombre del sitio oficial y posible autor (si es que lo hay). URL. Consultado el día:[Fecha de consulta]