Paradigmas de Programación

PEP 1
20 de Octubre de 2014

Rut: _____
Profesor: _____

Parte conceptual (**20 pts**)

1. **(2 pto)** Según el artículo "General features of imperative languages", indique que aseveración es **incorrecta**:
   a) Las declaraciones de control alteran el flujo normal de ejecución.
   b) Los llamados a funciones pueden ser por valor, por nombre o por referencia.
   c) Las variables pueden ser estáticas, dinámicas en stack, dinámicas implícitas en heap y dinámicas implícitas en heap.
   d) Las variables en los lenguajes imperativos son estáticamente tipadas.
   e) Todas las aseveraciones anteriores son correctas

2. **(2 pto)** ¿Qué aseveraciones son correctas con respecto al scope en C?
   I. El valor de una variable creada localmente en una función es imposible accederlo desde otras funciones que sean llamadas desde de la primera.
   II. Las variables cuyo contenido ha sido declarado utilizando memoria del heap son visibles por todo el programa.
   III. Las constantes y macros del preprocesador de C son visibles por todo el código del programa.
   IV. Es posible declarar una variable localmente con el mismo nombre que una variable global, lo cual oculta el acceso a la variable global.
   a) Sólo iv es correcta
   b) i y iv son correctas
   c) i, ii y iii son correctas
   d) ii, iii y iv son correctas
   e) Todas son correctas

3. **(2 pto)** ____ (Verdadero o falso) El uso de la palabra reservada "const" está relacionado con el proceso de binding de una variable (Justificar tanto si es verdadero como falso).

Falso, el uso de "const" está relacionado con la mutabilidad de una variable

4. **(2 pto)** Considere la siguiente situación. Se cuenta con una función en Scheme (f x) que realiza operaciones computacionales costosas a partir de la variable x entera. Dependiendo del valor real de x que es pasado a la función, ésta puede tardar desde segundos hasta horas para su ejecución. En la práctica se suelen estructurar los

llamados necesarios a la función f en una lista L, sin embargos los resultados de la función f no siempre son usados inmediatamente. Lo que si se suele hacer es determinar el largo de la lista, seleccionar llamados particulares, reordenar la lista de llamados, entre otras cosas. ¿De qué manera haría usted la implementación de la lista para estructurar los llamados de la función f, de manera que no requieran un uso excesivo de tiempo para operaciones que no requieren de los resultados otorgados por f? Además de responder su pregunta y justificarla, indique un ejemplo:

Evaluación perezosa (lazy evaluation) en vez de eager, ya que sólo generará la promesa de ejecución y no la ejecución directamente, a menos que ésta sea requerida de manera explícita.

(list (lazy (f 10)) (lazy (f 100)) (lazy (f 2)))

5. **(2 pto)** ¿Qué tipo de recursión se emplea en el siguiente código?

```
(define (f n m)
        (if (or (= n 0) (= m 0) (= n m))
                1
                (+ (f (- n 1) (- m 1)) (f (- n 1) m))
        )
)
```

_____

arbórea

6. **(2 pto) ____** (Verdadero o falso) los TDAs son propios de la programación funcional (Justificar tanto si es verdadero como falso).
   F, los TDAs son independientes del paradigma y lenguaje de programación.

7. **(2 pto)** Indicar las unidades de programación en los siguientes paradigmas:
   a) Imperativo-Procedural : _____ (procedimientos)
   b) Funcional: _____ (funciones)
   c) Lógico: _____ (relaciones) o (reglas y hechos))

8. **(2 pto)** Cuando se realiza un paso por referencia, cuál de las opciones son correctas:
   I) Se trabaja bajo las variables originales, pudiendo modificar estas.
   II) Se realiza una copia de la variable (no puntero).
   III) Se realiza una copia de la dirección de memoria de la variable.
   IV) Si cambio a donde apunta el puntero que fue pasado por parámetros, también cambia la dirección de la variable original.
   V) Si cambio donde apunta el puntero que fue pasado por parámetros, no se modifican los originales.

a) Sólo I
b) Sólo II
c) Sólo III

d) I, III, V
e) I, II y IV

9. **(2 pto)** ¿Qué paradigma de programación se ajusta mejor a las características de SQL? justifique (máximo en 2 línea)

_____

(lógico ya que SQL opera sobre relaciones (tablas) que tienen un paralelo con los conjuntos de hechos y reglas (relaciones) especificadas en lenguajes como prolog)

10. **(2 pto)** Frente a los siguientes hechos y relaciones en prolog que sirven para contar los elementos de una lista, indique qué variables se encuentran universalmente cuantificadas y cuáles son existencialmente cuantificadas.

```
contar([ ], C):-C = 0.
contar([H|T], C):-contar(T, Cold), C is Cold + 1.
```

"C" se encuentra universalmente cuantificada
"T" se encuentra universalmente cuantificada
"Cold" se encuentra existencialmente cuantificada
"H" se encuentra universalmente cuantificada
(Responder 1 da 0 puntos, responder 2 da 1 punto, responder 3 da 1 punto y responder 4 da los 2 puntos)

Desarrollo **(40 pts)**

1. **(14 pts.)** Hay una persona que ha decidido establecer su carpa en la playa, sin embargo se da cuenta que en ese lugar el mar se lleva arena por cada ola que llega a la playa; cada vez que llega una ola esta consume un área de playa de 10 m^2 más que la ola anterior, formando un semicírculo centrado en el mismo centro de la playa. Las olas llegan a la playa cada 3 minutos.

a) Realice un programa en C que entregue el tiempo en el cual será alcanzada por el agua la carpa si ésta se ubica 100 metros de la línea costera original y a 40 metros del centro de la playa, estos parámetros de entrada deben ser declarados como constantes del preprocesador. **(8 pts)**

Implementar solución utilizando además:

b) Bucles "lógicamente controlados" **(2 pts)**.

c) Al menos un procedimiento que haga uso de paso de parámetros por referencia **(2 pts)**.

d) Un mensaje por pantalla utilizando funciones de stdio con la respuesta al problema **(2 pts)**.

Solución:

```
#include <stdio.h>
#include <math.h>

#define TIEMPO_ENTRE_OLAS 3 //Trabajando el tiempo en minutos
#define INCR_AREA_CONSUMIDA_OLA 10 //Trabajando el área en metros cuadrados
#define X_CARPA 40 //Coordenada X de la carpa
#define Y_CARPA 100 //Coordenada Y de la carpa

void calculaRadio(float x, float y, float *res); //Calcula un pitágoras esta función

int main(int argc, char **argv) {
	int contTiempo = 0;
	float areaActualConsum = 0, area_extra_consumida = 0;
	calculaRadio(X_CARPA, Y_CARPA, &radioCarpa);
	float radioActual = 0;
	while (radioActual < radioCarpa ) {
		area_extra_consumida += INCR_AREA_CONSUMIDA_OLA;
		areaActualConsum += area_extra_consumida;
		radioActual = sqrt(areaActualConsum/M_PI);
		contTiempo += TIEMPO_ENTRE_OLAS;
	}
	printf("Las olas se llevaran la carpa luego de %d minutos\n", contTiempo);
	return 0;
}
```

//El resultado debe dar 255 minutos.

2. **(14 pts.)** Se provee de una implementación parcial del TDA matriz dispersa que permite almacenar sólo aquellos elementos cuyos valores son distintos de cero (en este contexto se interpretan como inexistentes). A saber, su representación consiste en una lista de listas. La lista principal o Matriz contiene N listas que representan las columnas. Cada una de estas listas está compuesta por un número entero seguido múltiples pares. El número entero representa el número de columna C, mientras que los pares sucesivos representan las filas en esa columna. Las filas son expresadas como pares de números enteros donde el primer elemento representa el número de fila F y el segundo elemento corresponde al ítem de la matriz contenido en la posición (C,F). El siguiente ejemplo ilustra la estructura antes descrita:

‘(‘(#columna ‘(#fila . elemento) … ‘(#fila . elemento)) … (#columna ‘(#fila . elemento) … ‘(#fila . elemento)))

De esta forma, la matriz:

	1	**2**	**3**	**4**
1	*0*	*0*	*2*	*0*
2	*0*	*3*	*8*	*0*
3	*0*	*7*	*0*	*0*
4	*0*	*0*	*0*	*9*

queda expresada como:

‘((2 ‘(2 . 3) ‘(3 . 7)) ‘(3 ‘(1 . 2) ‘(2 . 8)) ‘(4 ‘(4 . 9)))

La implementación parcial del TDA cuenta con los constructores MatrizDispersa, Columna y Fila para la construcción de matrices dispersas, columnas y filas respectivamente. De esta forma es posible crear la matriz anterior de la siguiente forma

(MatrizDispersa (Columna 2 (Fila 2 3) (Fila 3 7)) (Columna 3 (Fila 1 2) (Fila 2 8)) (Columna 4 (Fila 4 9)))

Los constructores realizan las validaciones correspondientes para respetar la estructura, tipos de datos y evitar duplicidad de filas y columnas dentro de una misma matriz.

Además de constructores se cuenta con funciones de pertenencia MatrizDispersa?, Columna? y Fila? para verificar que un valor es una matriz dispersa, columna y fila respectivamente.

Su labor es implementar en Scheme los siguientes selectores indicando la firma (signature) de cada función (esto es dominio recorrido) y el tipo de recursión empleada (si aplica).

(getElemento nCol nFil matriz) **(3 pts)** ;retorna el elemento contenido en la posición (nCol,nFil) de la matriz.
(getColumna nCol matriz) **(3 pts)** ;retorna la columna nCol de la matriz
(getNumCol col) **(1 pto)** ;retorna el número de la columna col
(getFila nFila col) **(3 pts)** ;retorna la fila contenida en la columna col
(getFilas col) **(2 pto)** ;retorna las filas donde existen elementos en la columna col
(getNumFila fil) **(1 pto)** ;retorna el número de la fila fil
(getElementoFila fil) **(1pto)** ;retorna el elemento contenido en la fila fil

Solución

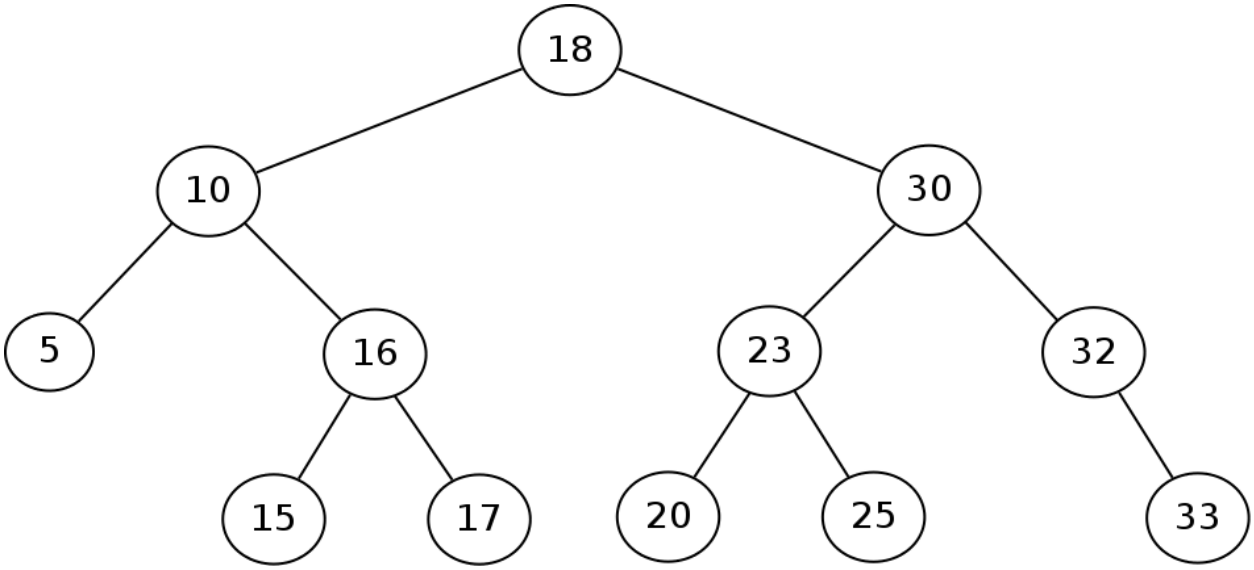
;SELECTORES

```
;Obtiene elemento específico en una matriz
;int x int x matriz -> int
(define (getElemento nCol nFil matriz)
        (getElementFila (getFila nFila (getColumna nCol matriz)))
)

;Obtiene la columna desde una matriz
;int x matriz -> columna
(define (getColumna nCol matriz)
   (if (and (MatrizD? matriz) (number? nCol))
        (if (null? matriz)
                null
                (if (= nCol (caar matriz))
                        (car matriz)
                        (getColumna nCol (cdr Matriz))
                )
        )
        #f
   )
)

;obtiene la Fila indicada en una columna
;int x columna -> fila
(define (getFila nFila col)
   (if (Columna? col)
        (if (null? col)
                0
                (if (= nFila (getNumFila (car col))
                        (car col)
                        (getFila nFila (cdr col))
                )
        )
        #f
```

```
    )
)

;obtiene la Fila indicada en una columna
;int x columna -> lista fila
(define (getFilas col)
    (if (Columna? col)
            (if (null? col)
                    null
                    (cdr col)
            )
            #f
    )
)

;Obtiene el numero de la fila
;fila -> int
(define (getNumFila fil)
            (if (Fila? fil)
                    (car fil)
                    #f
            )
)

;Obtiene elemento de la Fila
;fila -> int
(define (getElementoFila fil)
            (if (Fila? fil)
                    (cadr fil)
                    #f
            )
)

;obtiene el número de la columna
;columna -> int
(define (getNumCol col)
            (if (Columna? col)
                    (car col)
                    #f
            )
)
```

3. **(12 puntos)** Un AVL es un árbol binario ordenado balanceado, es decir, es un árbol que tiene dos hijos, los cuales los menores se encuentran a la izquierda la raíz y los mayores a la derecha de la raíz. Además, es un árbol balanceado, en donde la altura (es

el camino más largo de un nodo a una hoja) de los árboles del hijo izquierdo con el derecho no es mayor a 1. Para la siguiente figura, realizar las siguientes preguntas:



a) Los hechos del árbol dado. Utilizar la forma padre(Padre, Hijo). **(3 pts)**.

padre(18, 10).
padre(18, 30).
padre(10, 5).
padre(10, 16).
padre(30, 23).
padre(30, 32).
padre(16, 15).
padre(16, 17).
padre(23, 20).
padre(23, 25).
padre(32, 33).

b) Una regla que permita determinar si dos nodos X e Y son hermanos. **(1 pts)**.

hermano(X,Y):- padre(Z, X) , padre(Z, Y).

c) Una regla que permita saber si un nodo se encuentra a la derecha y una para saber si uno nodo está a la izquierda de otro, ambas reglas deben llamarse “derecha” e “izquierda” respectivamente. **(2 pts)**.

derecha(X,Y):- X>Y.
izquierda(X,Y):- X<Y.

d) Se necesita responder la siguiente pregunta: Dado el valor de un nodo, saber cuales son todos los antecesores de éste. Usar la forma: antecesor(X,Y) **(3 pts)**.

antecesor(X,Y):- padre(X,Y).
antecesor(X,Y):- padre(X,Z) , antecesor(Z,Y).

e) Se necesita responder la siguiente pregunta: Dado el valor de un nodo, saber cuales son todos los descendientes de éste. Usar la forma: descendiente (X,Y) **(3 pts)**.

descendiente(X,Y):- padre(Y,X).
descendiente(X,Y):- padre(Z, X) , descendiente(Z, Y).

PARA LA POR, mismo problema pero esta vez centrado en MODIFICADORES y ademas implementar funciones de pertenencia

(setElementoFila fil elemento) (2 pts) ;cambia el elemento contenido en la fila fil
(setNumFila fil nFil) (2 pts) ;cambia el numero de la fila fil por nFil
(setNumCol col nCol) (2 pts) ;cambia el numero de la columna col por nCol

;MODIFICADORES

```
(define (setElementoFila fil elemento)
        (if (and (Fila? fil) (not (null? fil)))
                (Fila (getNumFila fil) elemento)
                fil
        )
)

(define (setNumFila fil nFil)
        (if (and (Fila? fil) (not (null? fil)))
                (Fila nFil (getElementoFila fil))
                fil
        )
)

(define (setNumCol col nCol)
        (if (and (Columna? col) (not (null? col)))
                (Columna nCol (getFilas col))
                col
        )
)
```