Paradigmas de la Programación – Recuperatorio del Primer Parcial

21 de Junio de 2016

Apellido y Nombre: _____

1. **[20 pt.]** En Perl las variables pueden tener alcance estático (léxico) o dinámico, usando distintas palabras clave en el momento de su asignación. En el siguiente programa se ve cómo las palabras clave "`local`" y "`my`" tienen distintos efectos en la forma cómo las variables adquieren su valor. Diga cuál de estas dos palabras claves se usa para que el valor de la variable se asigne con alcance estática y cuál con alcance dinámico, y explique por qué.

```
sub visible {
    print "la variable tiene el valor $var\n";
}
sub uno {
    local $var = 'valor uno';
    visible();
}
sub dos {
    my $var = 'valor dos';
    visible();
}

$var = 'global';
visible();          # imprime "global"
uno();              # imprime "valor uno"
dos();              # imprime "global"
```

2. **[10 pt.]** Diagrame una secuencia de pilas de ejecución que representen los los diferentes momentos de la ejecución del siguiente programa, mostrando cómo se apilan y desapilan los diferentes *activation records* a medida que se va ejecutando el programa.

```
let fun g(y) = y+3
    fun h(z) = g(g(z))
in
    h(3)
end;
```

3. **[10 pt.]** Calcule el tipo de datos de la siguiente función en ML. Provea el árbol sintáctico de la función y aplique de forma explícita el algoritmo de inferencia de tipos, ya sea sobre el árbol mismo o como sistema de ecuaciones.

```
fun f(x, y, z) = x && (y or z)
```

4. **[20 pt.]** Java tiene una construcción lingüística llamada "`try... catch... finally...`" cuya semántica consiste en que el bloque de código bajo el alcance de `finally` se ejecuta siempre, tanto si se lanza una excepción bajo el alcance de `try` como si no se lanzó. Si se lanza una excepción, la excepción funciona exactamente igual que si no existiera el bloque "`finally`" (aunque el bloque

`finally` se ejecuta en cualquier caso), si se puede capturar mediante el `catch` se captura, y si no sigue buscando hasta que encuentra un `catch` que la pueda capturar. Sabiendo esto, explique qué se imprime si ejecutamos los siguientes tres programas y por qué.

```
class Ejemplo1 {
   public static void main(String args[]){
     try{
        System.out.println("primera sentencia del bloque try");
        int num=45/0;
        System.out.println(num);
     }
     catch(ArrayIndexOutOfBoundsException e){
        System.out.println("ArrayIndexOutOfBoundsException");
     }
     finally{
        System.out.println("bloque finally");
     }
     System.out.println("fuera del bloque try-catch-finally");
   }
}
```

```
class Ejemplo2 {
   public static void main(String args[]){
      try{
         System.out.println("primera sentencia del bloque try");
         int num=45/0;
         System.out.println(num);
      }
      catch(ArithmeticException e){
         System.out.println("ArithmeticException");
      }
      finally{
         System.out.println("bloque finally");
      }
      System.out.println("fuera del bloque try-catch-finally");
   }
}
```

```
class Ejemplo3 {
  public static void main(String args[]){
    try{
       System.out.println("primera sentencia del bloque try");
       int num=45/3;
       System.out.println(num);
    }
    catch(ArrayIndexOutOfBoundsException e){
       System.out.println("ArrayIndexOutOfBoundsException");
    }
    finally{
       System.out.println("bloque finally");
    }
    System.out.println("fuera del bloque try-catch-finally");
  }
}
```

5. **[20 pt.]** En muchos lenguajes de programación no se implementa herencia múltiple, sino que se usan mecanismos alternativos. En este ejemplo de Scala, explique cómo se usa la construcción `trait` de una forma parecida a las interfaces de Java o los mixins de Ruby para conseguir una expresividad parecida a la herencia múltiple de C++.

```
trait Cantante{
  def cantar { println("␣cantando␣...␣") }
  //mas metodos
}

class Persona{
  def explicar {  println ("␣humano␣") }
  //mas metodos
}

class Actor extends Persona with Cantante
class Actor extends Cantante with Performer
```

6. **[20 pt.]** En un lenguaje con pasaje por referencia, qué sentencias tenemos que incorporar para que una función dé los mismos resultados que obtendríamos si el pasaje de parámetros fuera por valor-resultado (*call by value-result*)? Por ejemplo, qué sentencias deberíamos añadir y cómo deberíamos modificar la siguiente función:

```
procedure p(var x, y : integer);

      begin

      ...

      x:=x+1;
      y:=y+1

      ...

      end;

p(z,z)
```