Paradigmas de la Programación – Examen Final

3 de Julio de 2015

Apellido y Nombre: _____

1. **[10 pt.]** Muestre con un ejemplo que la siguiente gramática es ambigua, y modifíquela para que se convierta en una gramática totalmente inambigua, sin ninguna ambigüedad.

```
<a> ::= <b> <c>
<a> ::= <b>
<b> ::= 'c' 'd' 'e'
<b> ::= 'c' 'd'
<b> ::= 'c'
<c> ::= 'd' 'e'
<c> ::= 'e'
```

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. Asuma que el lenguaje de programación tiene alcance estático.

   Nota de estilo: El nivel de granularidad de la descripción puede limitarse a bloques del texto del programa, no es necesario diagramar como estados distintos de la ejecución las instrucciones que no apilen a otro bloque. Si lo desea, puede usar índices en el mismo texto del programa para una representación más compacta de la pila.

```
var x=1;
var y=5;
function g(z) {return x+z;}
function f(y) {
  var x = y*y;
  return g(y);
}
var x=5;
f(2)
```

3. **[10 pt.]** En el siguiente programa,

```
begin
  integer n;
  procedure p(k: integer);
    begin
      k := k+2;
      print(n);
      n := n+(2*k);
    end;
  n := 4;
  p(n);
  print(n);
end;
```

Qué dos valores se imprimen si `k` se pasa...

a) por valor?

b) por valor-resultado?

c) por referencia?

4. **[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 a(x,y) = (x > 2) orelse (y < 10);
```

5. **[10 pt.]** En el siguiente código en Ruby, explique cómo se aproxima a la expresividad propia de un lenguaje con herencia múltiple, incluso si Ruby es un lenguaje que no tiene herencia múltiple sino simple. Compare este mecanismo con el de otro lenguaje que no tiene herencia múltiple sino simple pero que usa otro mecanismo para aproximarse a la expresividad de la herencia múltiple: Java con sus interfaces.

```
module EmailReporter
    def send_report
      # Send an email
    end
  end

  module PDFReporter
    def send_report
      # Write a PDF file
    end
  end

  class Person
  end

  class Employee < Person
    include EmailReporter
    include PDFReporter
  end

  class Vehicle
  end

  class Car < Vehicle
    include PDFReporter
    include EmailReporter
  end
```

6. **[10 pt.]** Señale cuáles de las siguientes expresiones son `true` en Prolog, es decir, en qué casos se encuentra una unificación exitosa para la expresión, y cómo se pueden instanciar los valores de las variables en las unificaciones exitosas.

a) `'Verdadero' = verdadero`

b) `Verdadero = verdadero`

c) `booleano(verdadero) = verdadero`

d) `booleano(verdadero,X) = booleano(Y,falso)`

e) `booleano(verdadero,X,falso) = booleano(Y,falso)`

f) `booleano(X) = X`

g) `bool(booleano(verdadero),booleano(falso)) = bool(X,Y)`

h) `bool(booleano(verdadero),X) = bool(X,booleano(falso))`

7. **[10 pt.]** Dibuje la jerarquía de clases en que se basa el siguiente código y explique la función de la palabra clave `virtual` en este caso.

```
class A {
public:
    explicit
    A( int z )
    : m_z( z )
    {}
    int Z() const {
       return m_z;
    }

    virtual void Save( std::ostream& ) const = 0;

private:
    int m_z;
};

class BA : public A {
public:
    BA( int z, int y )
    : A( z ), m_y( y )
    {}
    int Y() const {
      return m_y;
    }

    void Save( std::ostream& ) const;

private:
    int m_y;
};

class CA : public A {
public:
   CA ( int z, int x )
   : A( z ), m_x ( x )
   {}
   int X() const {
      return m_x;
   }
   void Save( std::ostream& ) const;

 private:
   int m_x;
 };
```

8. **[10 pt.]** Escriba en pseudocódigo dos versiones secuenciales (no concurrentes) del siguiente programa, una en paradigma funcional y otra en paradigma imperativo. Argumente las ventajas y desventajas de las tres versiones del programa usando los conceptos de *velocidad*, *overhead* y *determinismo*.

```
class Ejemplo extends RecursiveTask<Integer> {
  final int n;
  Ejemplo(int n) { this.n = n; }
  Integer compute() {
    if (n <= 1)
       return n;
    Ejemplo f1 = new Ejemplo(n - 1);
    f1.fork();
    Ejemplo f2 = new Ejemplo(n - 2);
    return f2.compute() + f1.join();
  }
}
```

9. **[20 pt.]** En las siguientes funciones en ML:

```
exception Excpt of int;
fun twice(f,x) = f(f(x)) handle Excpt(x) => x;
fun pred(x) = if x = 0 then raise Excpt(x) else x-1;
fun dumb(x) = raise Excpt(x);
fun smart(x) = 1 + pred(x) handle Excpt(x) => 1;
```

Cuál es el resultado de evaluar cada una de las siguientes expresiones?

a) `twice(pred,1)`

b) `twice(dumb,1)`

c) `twice(smart,1)`

Explique qué excepción se levanta en cada caso y dónde se levanta. Ayúdese de los respectivos diagramas de pilas de ejecución

Ejercicios para libres

1. **[-5 pt.]** El siguiente es un ejemplo de "*spaghetti code*". Reescríbalo en pseudocódigo de forma que NO use saltos (GOTO), y en cambio use programación estructurada en bloques.

```
10 i = 0
20 i = i + 1
30 PRINT i; " squared = "; i * i
40 IF i >= 10 THEN GOTO 60
50 GOTO 20
60 PRINT "Program Completed."
70 END
```

2. **[-5 pt.]** Si el resultado del siguiente programa es 19, qué tipo de alcance tiene el lenguaje de programación en el que está escrito, estático o dinámico?

```
val x = 4;
    fun f(y) = x*y;
    fun g(x) = let
        f(3) + x;
    g(7);
```

3. **[-5 pt.]** Identifique en el siguiente código en C++ un problema con la herencia del miembro `meow`.

```
class Felino {
public:
 void meow() = 0;
};

class Gato : public Felino {
public:
 void meow() { std::cout << "miau\n"; }
};

class Tigre : public Felino {
public:
 void meow() { std::cout << "ROARRRRRR\n"; }
};

class Ocelote : public Felino {
public:
 void meow() { std::cout << "roarrrrr\n"; }
};
```

4. **[-5 pt.]** En el siguiente código en Ruby, describa la visibilidad de la variable `cuenta`.

```
class Ser
    @@cuenta = 0

    def initialize
        @@cuenta += 1
        puts "creamos un ser"
    end
    def muestra_cuenta
        "Hay #{@@cuenta} seres"
    end
end
class Humano < Ser
   def initialize
       super
       puts "creamos un humano"
   end
end
class Animal < Ser
   def initialize
       super
       puts "creamos un animal"
   end
end
class Perro < Animal
   def initialize
       super
       puts "creamos un perro"
   end
end

Humano.new
d = Perro.new
puts d.muestra_cuenta
```