Paradigmas de la Programación – Primer Parcial

20 de Abril de 2017

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 inambigua.

```
<s> ::= <v> | <v> 'pip' <v> | <v> 'pop' <v>
<t> ::= 'a' | 'b' | 'c' | 'd'
<v> ::= <v> <t> | <t>
```

2. **[15 pt.]** Diagrame los sucesivos estados por los que pasa la pila de ejecución al ejecutar el siguiente programa, teniendo en cuenta que el lenguaje tiene alcance estático.

```
Program A()
{   x, y, z: integer;

    procedure B()
    {   y: integer;
        y=0;
        x=z+1;
        z=y+2;
    }

    procedure C()
    {   z: integer;

        procedure D()
        {   x: integer;
            x = z + 1;
            y = x + 1;
            call B();
        }
        z = 5;
        call D();
    }
    x = 10;
    y = 11;
    z = 12;
    call C();
    print x, y, z;
}
```

3. **[10 pt.]** En el programa del ejercicio anterior, ¿qué se imprimirá si el lenguaje tiene alcance estático? ¿Qué se imprimirá si el lenguaje tiene alcance dinámico?

4. **[15 pt.]** En el siguiente código en Java, explique qué sucede (cuáles son los cambios en los sucesivos estados de la máquina, a nivel de pila de ejecución) cuando trato de abrir un archivo que se existe y qué sucede cuando trato de abrir un archivo que no existe. Ayúdese de pilas de ejecución para ilustrar su explicación cuando lo crea conveniente. No es necesario diagramar todos los estados de la ejecución si no lo cree necesario, pero sí explicar detalladamente qué sucede en la ejecución del programa.

```
public static void cat(File named) {
    RandomAccessFile input = null;
    String line = null;

    try {
        input = new RandomAccessFile(named, "r");
        while ((line = input.readLine()) != null) {
            System.out.println(line);
        }
        return;
    } catch(FileNotFoundException fnf) {
        System.err.println("File: " + named + " not found.");
    } catch(Exception e) {
        System.err.println(e.toString());
    } finally {
        if (input != null) {
            try {
                input.close();
            } catch(IOException io) {
            }
        }
    }
}
```

5. **[10 pt.]** En Javascript no hay clases pero sí objetos. Vea cómo se instancian objetos en el siguiente código de Javascript y explique cuál sería la función de la palabra clave `constructor` estableciendo un paralelismo con lenguajes como C++ o Java.

```
function User (theName, theEmail) {
    this.name = theName;
    this.email = theEmail;
    this.currentScore = 0;
}

User.prototype = {
    constructor: User,
    saveScore:function (theScoreToAdd)  {
        this.quizScores.push(theScoreToAdd)
    },
    changeEmail:function (newEmail)  {
        this.email = newEmail;
        return "New Email Saved: " + this.email;
    }
}
```

6. **[10 pt.]** Para solucionar los *name clashes*, algunos lenguajes implementan heurísticas para decidir qué implementación seleccionar en el caso de un *name clash*. Explique qué estrategia utiliza Java para evitar este tipo de problema.

7. **[10 pt.]** Quisimos hacer un programa que convierta un arreglo de números en una cifra, por ejemplo, que convierta `[3,8,5]` en `385`. Hicimos dos versiones del mismo, las que se ven acá abajo:

```
def toNum(q: scala.collection.Queue1[Int]) = {
  var num = 0
  while (!q.isEmpty) {
    num *= 10
    num += q.dequeue
  }
  num
}
```

```
def toNum(q: scala.collection.Queue2[Int]) = {
  def recurse(qr: scala.collection.Queue2[Int], num: Int): Int = {
    if (qr.isEmpty)
      num
    else {
      val (digit, newQ) = qr.dequeue
      recurse(newQ, num * 10 + digit)
    }
  }
  recurse(q, 0)
}
```

Estas dos variantes, escritas en Scala, usan `var` y `val`, que son palabras claves para definir variables en Scala, una para definir variables mutables y otra para definir variables inmutables (que no pueden ser actualizadas) ¿Cuál de estas dos variantes tiene todas las propiedades de una componente declarativa, y cuál tiene alguna propiedad no declarativa?

8. **[10 pt.]** En el siguiente código, ¿puedo asegurar si este lenguaje tiene pasaje de parámetros por *call-by-value*, *call-by-reference*, *call-by-value-resultx*, *call-by-need* o *call-by-name*? ¿Qué tipos de pasaje por parámetros voy a poder distinguir según el output?

```
    begin
    array a[1..10] of integer;
    integer n;
    procedure p(b: integer);
        begin
        print(b);
        n := n+1;
        print(b);
        end;
    a[1] := 10;
    a[2] := 20;
    a[3] := 30;
    a[4] := 40;
    n := 1;
    p(a[n+2]);
    print(a);
    end;
```

9. **[10 pt.]** Las funciones virtuales de C++ se implementan mediante *dynamic dispatch*, es decir, seleccionando el código que se aplica en tiempo de ejecución. Partiendo de este hecho, explique por qué las funciones virtuales de C++ suponen más *overhead* que las funciones estáticas, y qué ventaja aportan.