Paradigmas de la Programación – Primer Parcial

21 de Abril de 2016

Apellido y Nombre: _____

1. **[5 pt.]** Seleccione todas las respuestas correctas entre las diferentes opciones que se dan para completar cada oración:

   a) Una gramática independiente de contexto (*context-free grammar*)...
      1) tiene predicados.
      2) es inambigua.
      3) tiene reglas de reescritura.
      4) describe la sintaxis de un lenguaje.
      5) es más expresiva que una gramática dependiente del contexto.
      6) es tan expresiva como una máquina de Turing.

   b) Las variables están compuestas de... (elija la opción más precisa)
      1) un nombre y un valor.
      2) un l-valor y un r-valor.
      3) un identificador y un valor.
      4) un identificador que está ligado a un valor que está ligado a una dirección en memoria.
      5) un identificador que está ligado a una dirección en memoria que puede contener un valor.
      6) una dirección de memoria, un valor y un identificador en el texto del programa.

   c) La clausura de una función es necesaria para...
      1) definir funciones en un alcance global
      2) mantener la referencia a las variables locales de la función si tenemos alcance dinámico
      3) mantener la referencia a las variables locales de la función si tenemos alcance estático
      4) mantener la referencia a las variables libres de la función si tenemos alcance dinámico
      5) mantener la referencia a las variables libres de la función si tenemos alcance estático

   d) La diferencia entre polimorfismo y sobrecarga es...
      1) que sobrecarga se aplica sólo a algunos tipos, mientras que polimorfismo es más general.
      2) que la sobrecarga se comprueba en tiempo de ejecución y el polimorfismo en tiempo de compilación.
      3) que en la sobrecarga tenemos diferentes implementaciones para un mismo símbolo y en el polimorfismo tenemos una sola implementación con tipos generales.
      4) que la sobrecarga usa tipos concretos y el polimorfismo usa variables de tipo.
      5) que la sobrecarga la analiza el compilador y el polimorfismo no.

   e) Las excepciones se pueden usar para optimizar código...
      1) porque tienen alcance dinámico.
      2) porque son saltos explícitos entre partes del programa.
      3) porque son imperativas.
      4) porque son funcionales.

2. **[10 pt.]** Muestre con un ejemplo que la siguiente gramática es ambigua, y modifíquela para que se convierta en una gramática inambigua.

```
<s> ::= <i> | <o>
<i> ::= 'i' <p> <s> | 'i' <p> <s> 'e' <s>
<p> ::= 'a' | 'b' | 'c' | 'd'
<o> ::= 'x' | 'y' | 'z' | 'w'
```

3. **[10 pt.]** En el siguiente programa, qué valores retornarán `foo` y `bar` si el alcance del lenguaje es dinámico? y si es estático?

```
const int b = 5;
int foo()
{
   int a = b + 5;
   return a;
}

int bar()
{
   int b = 2;
   return foo();
}

int main()
{
   foo();
   bar();
   return 0;
}
```

4. **[20 pt.]** En este fragmento de código se captura una excepción y se vuelve a lanzar la misma excepción. Explique qué sucede en este fragmento de código, ayudándose de un diagrama de una secuencia de estados de la pila de ejecución, mostrando cómo se apilan y desapilan los diferentes *activation records* a medida que se va ejecutando el programa. Para mayor claridad, puede acompañarlo de una descripción verbal. Describa también verbalmente cuál sería el objetivo de este programa. ¿Qué sentido tiene capturar una excepción para volver a lanzarla? ¿Es eso lo único que se hace?

```
ITransaccion transaccion = null;
try
{
 transaccion = sesion.EmpiezaTransaccion();
 // hacer algo
 transaccion.Commit();
}
catch
{
 if (transaccion != null) { transaccion.RestaurarEstadoPrevio(); }
 throw;
}
```

5. **[15 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)
```

6. **[20 pt.]** El lenguaje de programación Scala implementa como parte del lenguaje una heurística llamada "linearización", que permite determinar qué implementación se usará en un objeto que hereda una componente con el mismo nombre de diferentes clases. Explique por qué es necesario que el lenguaje implemente esta heurística (qué problema resuelve) y dé un ejemplo donde se muestre una configuración donde entraría en juego esta heurística para evitar el problema, por ejemplo, la configuración conocida como el problema diamante.

7. **[20 pt.]** Java implementa un tipo de pasaje de parámetros que consiste en pasar por valor la referencia a un objeto. Mediante este tipo de pasaje de parámetros, ¿Se puede implementar la función *swap* para que intercambie el objeto A por el objeto B? ¿Se puede implementar la función *swap* para intercambiar algún elemento, algún pedazo de software? ¿Cuál?