Paradigmas de la Programación – Recuperatorio del Segundo Parcial

15 de Junio de 2017

Apellido y Nombre: _____

1. **[12 pt.]** La siguiente expresión está mal tipada:

   ```
   f(x) = g(x)*2 > ( g(x) || true )
   ```

   Muestre con el árbol para inferencia de tipos dónde se encuentra el conflicto y en qué consiste. Cómo podría abordar este conflicto un lenguaje de tipado fuerte? y uno de tipado débil?

2. **[13 pt.]** Ante lenguajes de programación inseguros podemos tomar estrategias de programación defensiva o de programación ofensiva. De los dos códigos que encontramos abajo, explique cuál de los dos estaría aplicando una estrategia ofensiva, cuál una estrategia defensiva. Explique qué tipo de vulnerabiliad encontramos en estos códigos, si vulnerabilidad en el sistema de tipos o vulnerabilidad de memoria, y en qué consiste esta vulnerabilidad de forma genérica.

```
const char* trafficlight_colorname(enum trafficlight_color c) {
    switch (c) {
        case TRAFFICLIGHT_RED:    return "red";
        case TRAFFICLIGHT_YELLOW: return "yellow";
        case TRAFFICLIGHT_GREEN:  return "green";
    }
    return "black";
}
```

```
const char* trafficlight_colorname(enum trafficlight_color c) {
    switch (c) {
        case TRAFFICLIGHT_RED:    return "red";
        case TRAFFICLIGHT_YELLOW: return "yellow";
        case TRAFFICLIGHT_GREEN:  return "green";
    }
}
```

3. **[13 pt.]** Los frameworks proveen al programador con un *boilerplate*. Explique qué es un *boilerplate*, qué ventajas provee, qué desventajas en términos de *overhead*. En el código que sigue, explique por qué este es un ejemplo de *boilerplate* y desarrolle (implemente) cómo el programador podría usarlo para instanciar un caso particular.

```
<!DOCTYPE html>
<html>
   <head>
      <title></title>
   </head>
   <body> </body>
</html>
```

4. **[13 pt.]** El lenguaje D tiene diseño por contrato.

> *Contracts are assertions that must be true at specified points in a program. Contracts range from simple asserts to class invariants, function entry preconditions, function exit postconditions, and how they are affected by polymorphism. Typical documentation for code is either wrong, out of date, misleading, or absent. Contracts in many ways substitute for documentation, and since they cannot be ignored and are verified automatically, they have to be kept right and up to date.*

Reescriba el siguiente ejemplo en pseudocódigo imperativo, sustituyendo el contrato por código sin este tipo de abstracciones, de forma que se logre el mismo efecto que con el contrato en D.

```
byte *memcpy(byte *to, byte *from, unsigned nbytes)
    in
    {
        assert(to + nbytes < from || from + nbytes < to);
    }
    out(result)
    {
        assert(result == to);
        for (unsigned u = 0; u < nbytes; u++)
            assert(to[u] == from[u]);
    }
    body
    {
        while (nbytes--)
            to[nbytes] = from[nbytes];
        return to;
    }
```

5. **[5 pt.]** El siguiente código está dentro del modelo de actores. Señale en el código los mecanismos propios de actores y sírvase de ellos para explicar cómo los actores implementan concurrencia declarativa.

```
import akka.actor._

case object PingMessage
case object PongMessage
case object StartMessage
case object StopMessage

class Ping(pong: ActorRef) extends Actor {
  var count = 0
  def incrementAndPrint { count += 1; println("ping") }
  def receive = {
    case StartMessage =>
        incrementAndPrint
        pong ! PingMessage
    case PongMessage =>
        incrementAndPrint
        if (count > 99) {
          sender ! StopMessage
          println("ping stopped")
          context.stop(self)
        } else {
          sender ! PingMessage
```

```
        }
    }
}

class Pong extends Actor {
  def receive = {
    case PingMessage =>
        println("  pong")
        sender ! PongMessage
    case StopMessage =>
        println("pong stopped")
        context.stop(self)
  }
}

object PingPongTest extends App {
  val system = ActorSystem("PingPongSystem")
  val pong = system.actorOf(Props[Pong], name = "pong")
  val ping = system.actorOf(Props(new Ping(pong)), name = "ping")
  // start them going
  ping ! StartMessage
}
```

6. **[13 pt.]** En el siguiente lenguaje de programación, la palabra "`local`" se usa para declarar variables en un alcance, y el operador "←" se usa para ligar la variable de la derecha a la de la izquierda, de forma que el valor de la variable de la izquierda será una referencia a la de la derecha. Todas las variables se representan en la pila como punteros a una estructura de datos en el heap.

   Diagrame los diferentes estados por los que pasa la pila de ejecución y muestre en qué momento se puede recolectar cada variable. Señale también si hay casos de variables que podrían ser recolectadas antes de que queden sintácticamente disponibles para que el recolector de basura las recolecte.

```
{ local bli
  local bla
  { local ble
    local blu
    local bla
    bla <- ble
    { local blo
      ble <- blo
    }
    bli = 3
  }
}
```

7. **[13 pt.]** Dada la siguiente base de conocimiento en Prolog, grafique o explique verbalmente cuál sería la secuencia de predicados por los que se realizaría la búsqueda para verificar si es cierto `(valido(lidia,smith,(3,10,10)))`..

```
turno(celia,rivas,(6,30,8)).
turno(celia,zilvetti,(7,14,11)).
turno(tomas,rivas,(7,11,10)).
turno(tomas,perez,(8,11,10)).
turno(tomas,schuster,(9,11,10)).
turno(lidia,zilvetti,(7,14,10)).
```

```
turno(lidia,schuster,(9,11,11)).
turno(esteban,rivas,(7,1,9)).

especialidad(rivas,oftalmologia).
especialidad(smith,oftalmologia).
especialidad(zilvetti,ginecologoa).
especialidad(roman,ginecologia).
especialidad(perez,endocrinologia).
especialidad(schuster,clinico).

valido(Paciente,Medico,(Mes,Dia,Hora)) :-
  not((turno(Paciente,OtroMedico,_),
       especialidad(OtroMedico,X), especialidad(Medico,X))),
  not(turno(Paciente,_,(Mes,Dia,Hora))),
  not(turno(_,Medico,(Mes,Dia,Hora))).
```

8. **[13 pt.]** Comente el siguiente texto en el que se compara subtipado e interfaces, y explique por qué un la herencia y el subtipado son distintos y qué ventajas o desventajas puede tener separarlos o usarlos juntos.

Inheritance is about gaining attributes (and/or functionality) of super types. For example:

```
class Base {
    //interface with included definitions

}

class Derived inherits Base {
    //Add some additional functionality.
    //Reuse Base without having to explicitly forward
    //the functions in Base
}
```

Here, a Derived cannot be used where a Base is expected, but is able to act similarly to a Base, while adding behaviour or changing some aspect of Bases behaviour. Typically, Base would be a small helper class that provides both an interface and an implementation for some commonly desired functionality.

Subtype-polymorphism is about implementing an interface, and so being able to substitute different implementations of that interface at run-time:

```
class Interface {
    //some abstract interface, no definitions included
}

class Implementation implements Interface {
    //provide all the operations
    //required by the interface
}
```

Here, an Implementation can be used wherever an Interface is required, and different implementations can be substituted at run-time. The purpose is to allow code that uses Interface to be more widely useful.

Your confusion is justified. Java, C#, and C++ all conflate these two ideas into a single class hierarchy. However, the two concepts are not identical, and there do exist languages which separate the two.

9. **[5 pt.]** Ordene los siguientes fragmentos de código de más de scripting a menos de scripting, siempre justificando qué decisiones de diseño se evidencian en cada uno de estos códigos que los caracterizan como instancias de lenguajes del paradigma de scripting.

```
proc for {initCmd testExpr advanceCmd bodyScript} {
    uplevel 1 $initCmd
    set testCmd [list expr $testExpr]
    while {[uplevel 1 $testCmd]} {
        uplevel 1 $bodyScript
        uplevel 1 $advanceCmd
    }
}
```

```
use strict;
use warnings;
use IO::Handle;

my ( $remaining, $total );
$remaining = $total = shift(@ARGV);
STDOUT->autoflush(1);
while ( $remaining ) {
    printf ( "Remaining %s/%s \r", $remaining--, $total );
    sleep 1;
}
print "\n";
```

```
class classname : public superclassname {
 protected:
 // instance variables

 public:
 // Class (static) functions
 static void * classMethod1();
 static return_type classMethod2();
 static return_type classMethod3(param1_type param1_varName);

 // Instance (member) functions
 return_type instanceMethod1With1Parameter (param1_type param1_varName);
 return_type instanceMethod2With2Parameters (param1_type param1_varName, param2_ty
};
```

```
for jpg; do
    png="${jpg%.jpg}.png"
    echo converting "$jpg" ...
    if convert "$jpg" jpg.to.png ; then
        mv jpg.to.png "$png"
    else
        echo 'jpg2png: error: failed output saved in "jpg.to.png".' >&2
        exit 1
    fi
done
echo all conversions successful
exit 0
```