Paradigmas de la Programación – Segundo Parcial

14 de Junio de 2018

Apellido y Nombre: _____

1. **[10 pt.]** En el siguiente fragmento de código, diga qué se imprime si el pasaje de parámetros es por valor, por referencia o por valor-resultado.

```
z : integer;
procedure p (x:integer)
  begin
    x := x+1 ;
    z := z+2;
  end
  z := 1;
  p(z);
  write(z)
```

2. **[10 pt.]** Dada la siguiente base de conocimiento en Prolog, liste los subobjetivos que hay que satisfacer para verificar que el predicado `ama(amanda,amalia).` es cierto.

```
amable(amalia).
amable(amadeo).
amable(amanda).

conoce(amalia,amanda).
conoce(amalia,amadeo).

ama(X,Y) :- amable(Y), conoce(X,Y).
conoce(X,Y) :- conoce(Y,X).
```

3. **[10 pt.]** En el siguiente fragmento de código, indique qué escribiría la función `p` (en la instrucción `write(x)`) en un lenguaje con alcance estático y un lenguaje con alcance dinámico.

```
  procedure p;
     x: integer;
     procedure q;
        begin x := x+1 end;
     procedure r;
        x: integer;
        begin x := 1; q; write(x) end;
     begin
     x:= 2;
     r
     end;
```

4. **[10 pt.]** El siguiente es un ejemplo de programación defensiva. Identifique la función en la que se podrían insertar funcionalidades de canonicalización. Explique cómo se modificaría el manejo de excepciones si en lugar de usar programación defensiva usaramos programación ofensiva.

```
function calc() {
  var a = form().a;
  var b = form().b;
  try {
    return add(a, b);
  } catch (err) {
    showErrorMessage(err);
  }
}
```

5. **[10 pt.]** Para el siguiente código en Java, diagrame los sucesivos momentos por los que pasa la pila de ejecución, y explique en qué momento hay algún objeto disponible para recolectar y cuál.

```
class Test
{
    String obj_name;

    public Test(String obj_name)
    {
        this.obj_name = obj_name;
    }

    static void show()
    {
        Test t1 = new Test("t1");
        display();
    }

    static void display()
    {
        Test t2 = new Test("t2");
    }

    public static void main(String args[])
    {
        show();
    }
}
```

6. **[10 pt.]** En el siguiente código en un dialecto de Lisp, identifique propiedades de los lenguajes de scripting de dominio específico.

```
(defun my-split-window-func ()
  (interactive)
  (split-window-vertically)
  (set-window-buffer (next-window) (other-buffer)))

(global-set-key "\C-x2" 'my-split-window-func )
```

7. **[10 pt.]** De los siguientes fragmentos de React, ¿cuál es declarativo? En el que no es declarativo ¿Qué fenómenos podemos encontrar que no son declarativos? Vemos que uno de los fragmentos tiene la palabra clave "`map`". Explique, en función de su razonamiento anterior, qué tipo de programación concurrente se desarrolla con *map-reduce*.

```
function double (arr) {
  return arr.map((item) => item * 2)
}
```

```
function double (arr) {
  let results = []
  for (let i = 0; i < arr.length; i++){
    results.push(arr[i] * 2)
  }
  return results
}
```

8. **[10 pt.]** El siguiente código en Perl tiene manejo de excepciones. Identifique el operador que lanza las excepciones (equivalente a `raise` o `throw`), el operador que delimita el alcance donde se puede lanzar una excepción (equivalente a `try`) y explique cómo se manejan las excepciones en este programa, si con un manejador o bien con alguna otra construcción lingüística.

```
my ( $error, $fallo );
{
    local $@;
    $fallo = not eval {
        open(FILE, $file) || die "No se pudo abrir el archivo: $!";
        while (<FILE>) {
            process_line($_);
        }
        close(FILE) || die "No se pudo cerrar el archivo: $!";
        return 1;
    };
    $error = $@;
}

if ( $fallo ) {
    warn "encontramos el error: $error";
}
```

9. **[10 pt.]** Lea esta descripción sobre el lenguaje de programación *Elixir* y argumente para qué tipo de proyecto sería adecuado y para qué tipo de proyecto no sería necesario. Argumente su posición basándose en las características del lenguaje que se mencionan en este texto.

> *Elixir is a general-purpose, functional language designed for building scalable and maintainable applications.*
>
> *The language complies with the bytecode seen on the Erlang VM (also known as BEAM). Its syntax is often compared to the ever popular Ruby on Rails development framework.*
>
> *Elixir code runs inside lightweight, isolated processes, which allows for thousands of processes to run concurrently in the same machine. This in turn allows for vertical scaling and uses all of a machine's resources as efficiently as possible.*
>
> *These processes are also able to communicate with other processes running on different machines in the same network, providing a solid foundation for distribution and allowing for horizontal scaling.*

Running into issues with running software is inevitable, but Elixir's fault-tolerant system can make the process a little less painful. It provides 'supervisors' that you can program with descriptions of how to restart certain parts of a system when things fail.

These parts will then revert to a 'known, initial state', which is guaranteed to work.

10. **[10 pt.]** El siguiente fragmento de código en Scala, identifique en el código por lo menos dos características propias de la programación orientada a actores y explíquelas. Explique, además, para qué serviría la línea "`import context`" y su relación con la programación declarativa o imperativa, según lo crea adecuado.

```
class Chopstick extends Actor {

  val log = Logging(context.system, this)

  import context._

  //When a Chopstick is taken by a hakker
  //It will refuse to be taken by other hakkers
  //But the owning hakker can put it back
  def takenBy(hakker: ActorRef): Receive = {
    case Take(otherHakker) =>
      otherHakker ! Busy(self)
    case Put(`hakker`) =>
      become(available)
  }

  //When a Chopstick is available, it can be taken by a hakker
  def available: Receive = {
    case Take(hakker) =>
      log.info(self.path + " is taken by " + hakker)
      become(takenBy(hakker))
      hakker ! Taken(self)
  }

  //A Chopstick begins its existence as available
  def receive = available
}
```