Paradigmas de la Programación – Examen Final

5 de Julio de 2018

Apellido y Nombre: _____

1. **[10 pt.]** La siguiente expresión está mal tipada:

```
f (g, x) = g(x) > x(g)
```

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

2. **[10 pt.]** En el siguiente programa, diagrame los diferentes estados por los que va pasando la pila de ejecución, con atención a las relaciones entre las variables.

```
#include < stdio.h >
int x=0;
void p(int,int);
void main(){
   int x = 1;
   p(x,x);
}
void p(int y, int z){
   x = x+1;
   y = y+1;
   z = z+1;
   printf("%d\n",x+y+z);
}
void p(int &y, int &z){
   x = x+1;
   y = y+1;
   z = z+1;
   printf("%d\n",x+y+z);
}
```

3. **[10 pt.]** Diagrame los sucesivos estados por los que pasa la pila de ejecución al ejecutar el siguiente programa 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;
twice(pred,1);
```

4. **[15 pt.]** El siguiente fragmento de código funciona porque se aplica un tipo de polimorfismo que recurre al subtipado que se articula a través de la herencia. En este lenguaje, el subtipado permite que una función se defina para tomar objetos de tipo T pero la función también funciona correctamente si se le pasan objetos de tipo S, siempre que S sea subtipo de T. Esto es así porque se aplica el principio de Liskov, enunciado por Bárbara Liskov y Jeannette Wing de la siguiente forma:

   > *Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.*

   Explique con sus propias palabras por qué funciona este código, recurriendo a los conceptos de polimorfismo, subtipado y herencia.

```
abstract class Animal {
    abstract String talk();
}

class Cat extends Animal {
    String talk() {
        return "Meow!";
    }
}

class Dog extends Animal {
    String talk() {
        return "Woof!";
    }
}

static void letsHear(final Animal a) {
    println(a.talk());
}

static void main(String[] args) {
    letsHear(new Cat());
    letsHear(new Dog());
}
```

5. **[10 pt.]** Si una componente de software usa una variable global en una guarda (como parte de una condición con "if"), entonces no es independiente de contexto. Si, en cambio, una componente de software modifica una variable global pero ésta no afecta a su ejecución, conserva la transparencia referencial pero puede tener efectos secundarios.

   Escriba dos programas en pseudocódigo, uno en el que se de el primer fenómeno (una variable global que hace que un programa no sea independiente de contexto, es decir, que los resultados de su ejecución no dependan solamente de sus parámetros de entrada), y otro en el que se de el segundo fenómeno (un programa que tiene efectos secundarios a través de una variable global).

6. **[10 pt.]** De la misma forma que las variables globales, el pasaje de variables por referencia puede producir efectos secundarios. Sin embargo, en componentes de software declarativas no hay efectos secundarios, y no hay diferencia entre pasaje de parámetros por valor o por referencia. Explique cómo se puede dar este fenómeno, y cuál es la restricción lingüística que imponen los lenguajes que se inscriben en el paradigma funcional para forzar esta propiedad en los programas que se escriben en esos lenguajes, es decir, para forzar que el pasaje de parámetros no pueda ser una vía para propagar efectos secundarios.

7. **[10 pt.]** Estos dos fragmentos de código tienen la misma semántica, pero en uno hay inversión de control e inyección de dependencia, y en el otro no. Explique cuál es cuál. En el que tiene inversión de control, identifique el hot spot donde se puede parametrizar la dependencia que se inyecta en lugar de que esté incrustada en el código.

```
class EventLogWriter //Escribir al log de eventos
{
    public void Write(string message) { }
}

class AppPoolWatcher
{
    // Handle para el escritor de eventos
    EventLogWriter writer = null;

    // Se llama a esta funcion cuando la pool de app tiene un problema
    public void Notify(string message)
    {
        if (writer == null)
        {
            writer = new EventLogWriter();
        }
        writer.Write(message);
    }
}
```

```
public interface INofificationAction
{
    public void ActOnNotification(string message);
}

class AppPoolWatcher
{
    // Handle para el escritor de eventos
    INofificationAction action = null;

    // Se llama a esta funcion cuando la pool de app tiene un problema
    public void Notify(string message)
    {
        if (action == null)
        {
            // Mapear la interfaz a una clase concreta
        }
        action.ActOnNotification(message);
    }
}

class EventLogWriter : INofificationAction // Escribir al log de eventos
{
    public void ActOnNotification(string message) { }
}
```

8. **[10 pt.]** Dada la siguiente base de conocimiento en Prolog, liste los subobjetivos que hay que satisfacer para verificar si el predicado `tio(Pedro,Juan).` es cierto o no.

```
hermano(Pedro, Julia).
hermana(Ana, Teresa).
hermano(Pedro, Esteban).
madre(Ana, Juan).
padre(Esteban, Juan).

tio(X,Y) :- hermano(X,Z), progenitor(Z,Y).
progenitor(X,Y) :- padre(X,Y).
progenitor(X,Y) :- madre(X,Y).
```

9. **[10 pt.]** En el siguiente programa en Erlang, identifique los fragmentos de programa con semántica concurrente y explique cuál es su semántica y por qué es concurrente, es decir, qué semántica expresan que no está incluída en la expresividad de las máquinas de Turing.

```
max(N) ->
    Max = erlang:system_info(process_limit),
    io:format("Maximum allowed processes:~p~n" ,[Max]),

    statistics(runtime),
    statistics(wall_clock),

    L = for(1, N, fun() -> spawn(fun() -> wait() end) end),
    {_, Time1} = statistics(runtime),
    {_, Time2} = statistics(wall_clock), lists:foreach(fun(Pid) -> Pid ! die end, L

    U1 = Time1 * 1000 / N,
    U2 = Time2 * 1000 / N,
    io:format("Process spawn time=~p (~p) microseconds~n" , [U1, U2]).
    wait() ->

    receive
       die -> void
    end.

for(N, N, F) -> [F()];
for(I, N, F) -> [F()|for(I+1, N, F)].

start()->
    max(1000),
    max(100000).
```

10. **[10 pt.]** En el siguiente programa en AWK, identifique por lo menos dos características propias de los lenguajes de scripting y justifíquelas con porciones del código.

```
    if ((x=index($1,"@")) > 0) {
          username = substr($1,1,x-1);
          hostname = substr($1,x+1,length($1));
          printf("username = %s, hostname = %s\n", username, hostname);
     }
```

Ejercicios para libres

1. **[-10 pt.]** El siguiente código (resumido) de un driver SCSI para Linux es spaghetti. ¿Por qué decimos que es spaghetti, cuál es la principal diferencia con el código estructurado? ¿Si el lenguaje obliga a usar código estructurado, qué estructura de datos puede usar el compilador para manejar la memoria de forma más eficiente? Reescriba el driver en pseudocódigo para que sea estructurado y no spaghetti.

```
wait_nomsg:
        if ((inb(tmport) & 0x04) != 0) {
                goto wait_nomsg;
        }
        ...
        for (n = 0; n < 0x30000; n++) {
                if ((inb(tmport) & 0x80) != 0) {
                        goto wait_io;
                }
        }
        goto TCM_SYNC;
wait_io:
        for (n = 0; n < 0x30000; n++) {
                if ((inb(tmport) & 0x81) == 0x0081) {
                        goto wait_io1;
                }
        }
        goto TCM_SYNC;
wait_io1:
        ...
TCM_SYNC:
        ...
```

2. **[-20 pt.]** En el siguiente código en C++, hay un problema con la herencia del miembro "meow", que viene dada por el tipo de herencia por defecto que tiene C++. Explique por qué C++ tiene herencia no virtual, sin reescritura, por defecto, y cómo habría que hacer para que este programa funcionara sin problemas.

```
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"; }
};
```