Paradigmas de la Programación – Segundo Parcial

14 de Junio de 2016

Apellido y Nombre: _____

1. **[20 pt.]** En el siguiente programa, identifique las porciones del programa que no son funcionales, y mencione por qué no lo son. Escriba en pseudocódigo un programa funcional con la misma semántica que este programa.

```
Module Modulo1
    Dim texto As String = "textoUno"

    Public Sub ConcatenaConGuion(ByVal textoAConcatenar As String)
        texto = texto & "-" & textoAConcatenar
    End Sub

    Sub Main()
        ConcatenaConGuion("textoDos")
    End Sub
End Module
```

2. **[10 pt.]** En el siguiente programa en Rust, identifique porciones del programa que tienen semántica puramente concurrente. Las porciones del programa que ha señalado, ¿son construcciones lingüísticas para forzar atomicidad en porciones del código?

```
fn main() {
    let greeting = "Hello";

    let mut threads = Vec::new();

    for num in 0..10 {
        threads.push(thread::spawn(move || {
            println!("{} from thread number {}", greeting, num);
        }));
    }

    for thread in threads {
        thread.join().unwrap();
    }
}
```

3. **[10 pt.]** Explique el rol del Security Manager en Java.

4. **[10 pt.]** Los actores (modelo de programación de concurrencia funcional) pueden realizar una acción que consiste en redefinirse a sí mismos, es decir, cambiar el código que ejecutan. Explique cómo esto se relaciona con el cambio de estado de un objeto en programación orientada a objetos, y por qué los actores no recurren al concepto de estado que usan los objetos.

5. **[10 pt.]** Dada la siguiente base de conocimiento en Prolog, explique cómo sería la sucesión de objetivos que hay que satisfacer para contestar la consulta `nieto(X,leoncio).`

```
progenitor(juan,alberto).
progenitor(luis,alberto).
progenitor(alberto,leoncio).
progenitor(geronimo,leoncio).
progenitor(luisa,geronimo).

hermano(A,B) :-
  progenitor(A,P),
  progenitor(B,P),
  A \== B.

nieto(A,B) :-
  progenitor(A,P),
  progenitor(P,B).
```

6. **[10 pt.]** En lenguajes inseguros, se pueden implementar políticas de seguridad llamadas "ejecución defensiva" o "programación defensiva". Describa algunas de estas políticas, por ejemplo, para pervenirse de vulnerabilidades por debilidad en el sistema de tipos de un lenguaje de scripting.

7. **[20 pt.]** A partir del siguiente programa en pseudocódigo, diagrame los diferentes momentos por los que pasa la pila de ejecución, y señale en esos momentos qué variables pueden ser recolectadas por el recolector de basura (*garbage collector*). En los *activation records* sólo necesita diagramar las variables locales. Asuma que todas las variables se representan mediante punteros en los *activation records*, y que el valor de la variable se encuentra en el heap, por lo tanto necesitan ser recolectadas por el recolector de basura. Explique si la memoria asociada a alguna variable podría haber sido liberada antes de que el recolector de basura la libere, por ejemplo, manualmente.

```
int bla
{ int bli = ...
  int blu = ...
  int blo = ...
  {
    int ble = bli * bli
  }
  bli = ...
  {
    int ble = blo + blo
  }
}
```

8. **[10 pt.]** Ordene los siguientes fragmentos de código de más de scripting a menos de scripting, y explique cuáles son los principios que han contribuído a su ordenamiento.

```
      PROGRAM HELLO
        DO 10, I=1,10
        PRINT *,'Hello World'
   10 CONTINUE
        STOP
        END
```

```
tell application "Scriptable Text Editor"
        make new window
        activate
        set contents of window 1 to "Hello World!" & return
end tell
```

```
print "Hello World!"
```

```
import std.stdio;
int main() { writefln("Hello world!"); return 0; }
```